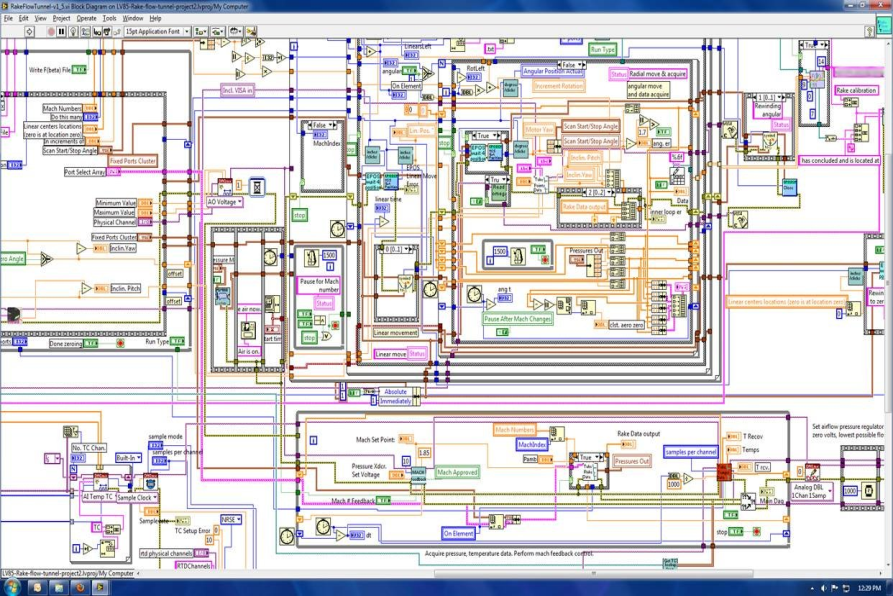
Custom Controls mit JFX 8

# TODO Motivation - Warum Custom Controls?

**Problem:** Existierender Code lässt sich häufig nicht 1:1 übernehmen!



**Ursache: Komplexität der Implementierung**

* kein einheitliches Vorgehen (MVC, MVVM, …)
* Kapselung / Sichtbarkeit verletzt
* implizite / versteckte Annahmen im Code
* fehlende Abstraktion / lose Kopplung

**Folge:** Anpassungen auf Code-Ebene notwendig > Code-Verständnis braucht Zeit

**Ziel:** Vereinfachung der Wiederverwendung

[Bild: SceneBuilder in Action]

**UseCases:**

* Wiederverwendung häufig genutzter Controls im SceneBuilder per Drag’n’Drop
* Verkauf von Custom Controls

**Wie? ⬄ Roadmap zum Vortrag:**

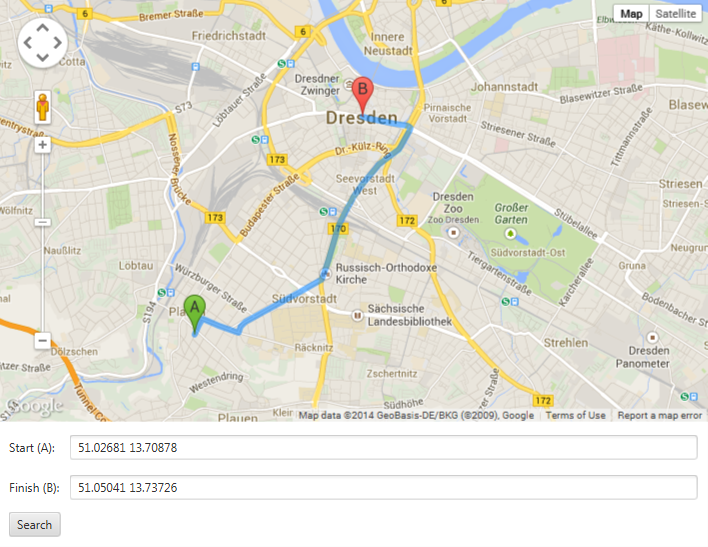
* Umsetzung einer Klassenbibliothek für wiederverwendbare UI-Komponenten unter Einhaltung der API zur Umsetzung von UI Controls mit JavaFX
* UI Patterns
* Architektur und Umsetzung von UI Controls
  + Styling mit CSS und Pseudoklassen
  + Skins
  + Erstellung eigener Controls
  + Testen von Controls

## TODO Demo: Checkbox

Bild | Diagramm | Philosophieren

* eigene CSS einbinden

## TODO Demo: Router / JFXMaps

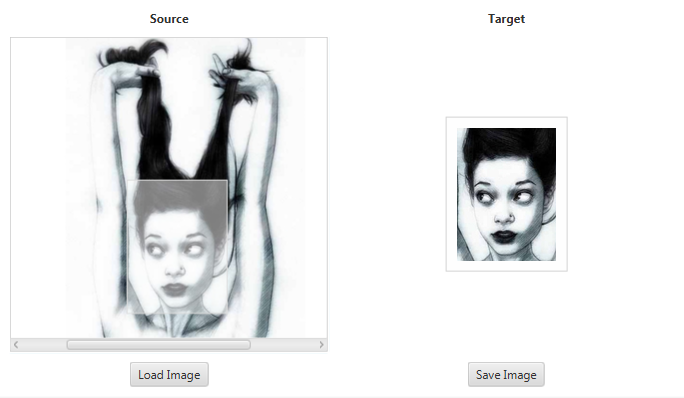


TODO nach Refactorings: Diagramm

**Features:**

* intend: navigation from origin to destination
* properties: source and target position
* integration of Google's Maps Javascript API
* customization of JavaFX WebView and aggregation of the result inside a custom control to be imported into the SceneBuilder
* batch property changes within a delay of 3 seconds using executor services on tasks

## TODO Demo: ImageCropper



TODO: Diagramm

 **intent:** load, crop and save images via moving a cropper rectangle

** properties:**

* source + target image
* cropper height + width
* cropper fill + stroke color

 custom control whose skin aggregates standard components as well as further custom components (source and target image view)

* source image view: offers object property 4 cropper rectangle

## TODO Demo: Master Details View

Bild | Diagramm | Philosophieren

## TODO Demo: TreeView

Bild | Diagramm | Philosophieren

# UI Patterns

* Patterns verstehen:
  + Anliegen:
    - **Wofür** ist es geeignet?
    - Welches Problem kann damit gelöst werden? (**Use Cases** / Best Practices aus der Praxis)
  + Implementierung:
    - **Wie** kann das Pattern umgesetzt werden?
    - Was ist der Kern? Was kann ich weglassen?
    - Welche Rollen sind beteiligt? (Produkt, Client, Fabrik, usw.)
  + Relationen zu anderen Patterns
* Sammlungen:
  + <http://ui-patterns.com/patterns>
  + <http://www.welie.com/patterns/>
  + uvvm.

## Architektur vs. Gestaltung

* Ziel: Umgang mit UI Elementen
  + Architektonischer Ansatz:
    - **Patterns:** MVC, MVVM, MVP
    - **Anliegen:** 
      * **Wofür:** Innen + Außen
        + Systeme als Ganzes planvoll Entwerfen und **Gestalten**
        + Benutzbarkeit ist nur ein Teilaspekt
        + Wie ist das System umgesetzt?
      * **Use Cases:** vielschichtige / komplexe / verteilte Geschäftsanwendungen als Ganzes (UI, AL, BL, Model)
    - **Implementierung:** siehe MVC vs. MVVM
  + Gestalterischer Ansatz:
    - **Patterns:** Master Details View, Ribbon (Multi-Line-Toolbar), GridData, Wizard, (Checkbox)
    - **Anliegen:**
      * **Wofür:** Außen
        + Entwicklung von Best Practices für die Darstellung + Interaktion eines Systems für den Benutzer
        + Ästhetik, Erscheinungsbild, Wahrnehmung, Benutzbarkeit beeinflussen
        + Teilgebiet der Architektur: ( Außen ) -> { SW-Systeme: UI }
      * **Use Cases:**
        + Reusable UI Components / UI Libraries

## TODO Aspekt Architektur - Oracle’s MVC Light vs. MVVM

TODO: Role Object Model für MVC und MVVM gegenüberstellen

## Aspekt Gestaltung – Control vs. Layout

* Layout > Darstellung
  + Theorie:
    - Dimension: 1 (Listen) | n (Matrizen, Gitter)
    - Ausrichtung: horizontal | vertikal
  + Praxis: Grid, AnchorPane, List, HBox, VBox, usw.
* Control > Interaktion
  + Theorie:
    - Paradigma: Aktion > Reaktion | Befehl > Aktion
  + Praxis: Button, Hyperlinks, Icons, DropDowns, PopUps, Ribbons, Wizard (?), ScrollPane, Accordeon, usw.

## Aspekt Einfachheit – Innen && Außen

“Try to make it as simple as possible but not simpler.” (A. Einstein)

* **Leitmotiv(f):** Minimalismus, den chaotische Komplexität ist abschreckend
* Einfachheit durch ..
  + Verstecken / Verbergen (unnötiger Details)
  + Ordnen / Organisieren / Gruppieren
* Patterns:
  + Schrittweise Offenlegung (Progressive Disclosure)
    - Verbergen von für den jeweiligen Anwendungsfall nicht essenziellen UI-Elementen
    - Anzeige bei Bedarf (interaktiv bei bestimmten Aktionen des Anwenders | automatisch in bestimmten Zeitintervall oä.)
  + Responsive Design
    - Abhängigkeit von Darstellung und Interaktion vom Kontext (Darstellungsmedium, Benutzereingaben)
    - Layout variabel
  + Responsive Enabling
    - deaktivierte Bereiche, die in Abhängigkeit des Kontextes aktiviert werden
    - Layout konstant

## CheatSheet: Building UI Controls in General

### Fachkonzept:

* Welche Informationen sollen Anwender warum sehen?
* Was sollen Anwender warum machen können?
* Wie hilft dies den Anwendern, ihre Aufgabe zu erfüllen?

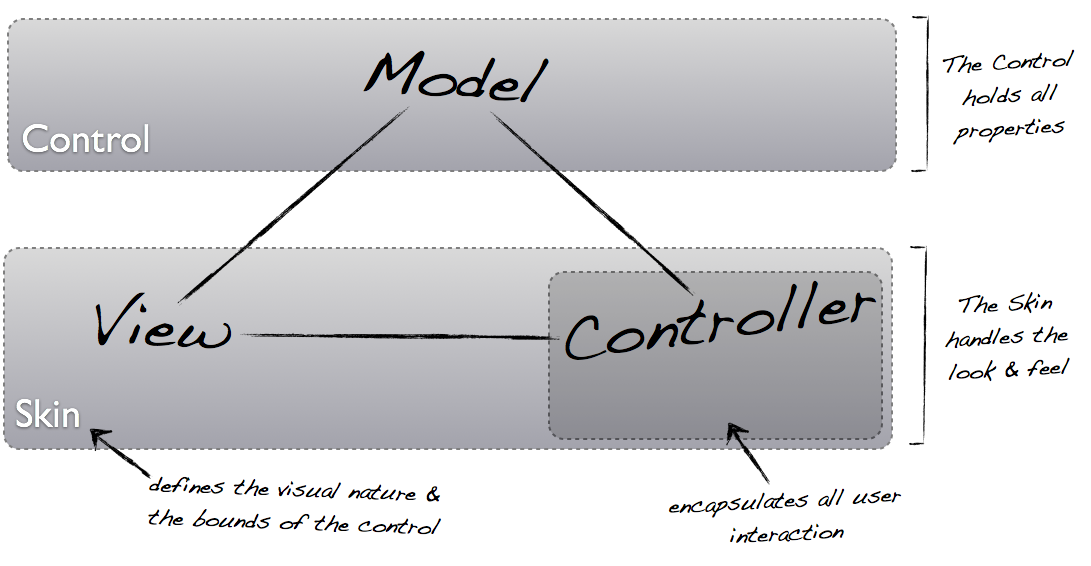
### Darstellung von Daten:

* Was ist der **Charakter / Metapher** passt zu einer darzustellenden Information? Welche Darstellungen lassen sich davon ableiten?
* Gibt es Wege, die die GUI zu **vereinfachen**?

[[Quelle]](http://www.roland-weigelt.de/downloads/misc/20120425_UI_Patterns.pdf), [[Quelle]](http://www.dlsi.ua.es/~santi/papers/UIConceptualPatterns.pdf), [[Quelle]](http://designingwebinterfaces.com/designing-web-interfaces-12-screen-patterns)

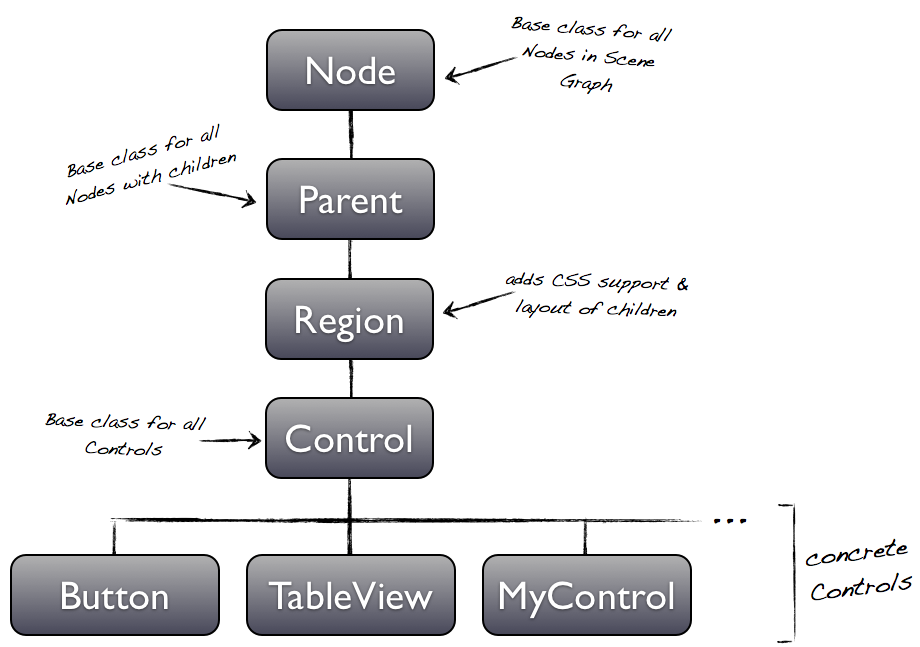
# Architektur von UI Controls

* JavaFX Controls folgen MVC



## Control (Model)

* kapselt den Zustand sowie entsprechende Funktionen zur Manipulation des Zustands (z.B. Definition von Properties bei reaktivem Programmierstil)
* keine Kenntnis über Layouting / Styling / Rendering oder UI-Interaktionen



* jede Control hat eine Referenz auf sein Skin zur Delegation des Layoutings
  + z.B. Berechnung Min-, Max- und Pref-Sizes in getMinWidth, getMinHeight, getPrefWidth, getPrefHeight, getMaxWidth, and getMaxHeight
  + ABER: baseLineOffset (Basislinie von Texten) wird an Node delegiert
* jede Änderung des Skins eines Controls markiert das Control so, dass es bei dem nächsten Layouting neu gerendert wird
  + Manuelles Triggern des Layoutings: Control.requestLayout()
* Stylen des (optionalen) Tooltipps eines Controls via CSS

## Skin (View, Controller)

* definiert Styling / Rendering des Controls (auch als Folge von UI-Interaktionen)
  + **ACHTUNG:** In den Methoden zum Layout-Computing dürfen keine rechenaufwendigen Operationen ausgeführt werden, da diese häufig aufgerufen werden.
* Implementierung der Antwort auf relevante Key-Events

#### JavaFX 2.2 vs. JavaFX 8

|  |  |  |
| --- | --- | --- |
|  | JavaFX 2.2 | JavaFX 8 |
| SkinBase API | restriktiv / intern | öffentlich (ohne Referenz auf Behavior API) |
| Behavior API | restriktiv / intern | restriktiv / intern (Ursache: Komplexität der Abbildung der Eingabe auf externe Aktionen) |
| Vererbungshierarchie | SkinBase und das Control bilden zwei gespiegelte Control-Hierarchien (Folgen: hohe Speicherkosten, SkinBase hatte dieselbe ID wie das Control selbst > Probleme beim Testen mit TestFX, Unverständliches Mirroring zwischen Controls) | SkinBase extends Object + Delegation an Control (z.B. SkinBase.getChildren()) |
| Referenzierung von Default Skins | Ausschließlich via CSS-Pseudo-Klasse –fx-skin | Programmatisch via Override Control.createDefaultSkin() + Java-Instanziierung (ABER: in Java-1.8.0-ea noch nicht enthalten) |
| SkinBase.layoutChildren(..) | Attribute x, y, width und height eines Controls mussten händisch kalkuliert warden > fehleranfällig | Attribute x, y, width und height eines Controls werden als Parameter übergeben |

## TODO Behavior

* Behandlung von Mouse und Key-Events (z.B. Definition Key-Bindings)
* Wie ist das Vorgehen in JavaFX 8?

[[Quelle]](https://wiki.openjdk.java.net/display/OpenJFX/UI+Controls+Architecture)

# Building Custom Controls

## Existierende Bibliotheken für Custom Controls (JavaFX)

(siehe Vergleichstabelle)

## Lebenszyklus von UI Controls

### Phasen

Folgende Phasen werden im Lebenszyklus eines UI Controls durchlaufen:

* 1. **Instanziierung des UI Controls:** new Button()
  2. **Setzen der „Default CSS Style Class“** im Konstruktor des Controls: getStyleclass().add("button")
     1. Jeder Control ist eine CSS Style Class zugeordnet, über die das Control „gestyled“ werden kann.
     2. **Ergebnis:**
        + Control mit 0er Höhe und Breite
        + Skin wird nicht instanziiert; keine Layout-Berechnungen
  3. **Platzierung des Control im Scenegraph:** getChildren().add(button)
  4. **Instanziierung des Skins** 
     1. **Definition via CSS:** Control.getUserAgentStylesheet():String
        + Auslesen der -fx-skin CSS Property zur Referenzierung des Skins (in Control.loadSkinClass())
        + Reflection
          - **Nachteil:** Refactoring
          - **Vorteil:** Aspekt Orientierung;
     2. **Programmatische Instanziierung im Control:** Control.createDefaultSkin():SkinBase
        + **Vorteil:** Keine Reflection
     3. **Keine Instanziierung des Skins**
        + Control funktional (ohne Layouting) einsetzbar
        + Fehlermeldung wird geloggt

### Disposal of UI Controls

**Use Case:** Unregister ChangeListeners defined as Lambdas

* JavaFX bietet gegenwärtig ausschließlich einen entsprechenden Hook via Skin.dispose(), eine entsprechende Methode für Controls ist allerdings geplant. Für die Zukunft ist allerdings auch dies geplant.

[[Quelle]](https://wiki.openjdk.java.net/display/OpenJFX/UI+Controls+Architecture)

## Styling mit CSS und Pseudo-Klassen

### Default Style Sheets / JavaFX Themes

* definiert Styles für Standard Controls von JavaFX
* extract from jfxrt.jar
* FX2: Caspian.css
* FX8: Modena.css

### Eigene CSS einbinden

Scene scene = new Scene(new Group(), 500, 400);

scene.getStylesheets().add("*path*/*stylesheet*.css");

### ID Styles

**Selektor:** #

**Implementierung:**

*FXML*

<ImageView id=”logoImageView” style="-fx-background-color: white;">

*Programmatisch*

### ImageView.getStylesheets().add("-fx-background-color: white;");

### Class Styles

**Selektor:** **.**

**Implementierung:**

*FXML*

<ImageView id=”logoImageView” styleClass="my-image-view">

*Programmatisch*

checkbox.getStyleClass().add("checkbox");

### Pseudo-Klassen

#### Existierende Pseudoklassen

* hover

HIER WEITER

#### Definition eigener Pseudoklassen

### TODO Variablen

### TODO Gradienten

### TODO SVG

[[Quelle: Oracle Tutorial]](http://docs.oracle.com/javafx/2/css_tutorial/jfxpub-css_tutorial.htm) [[Quelle: Oracle JavaFX CSS Reference Guide (JavaFX 2)]](http://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html)

## Austausch des Skins

### Use Cases

* Fixed Ratio Control (f.e. 16:9)

### Zuordnung des Skins zum Control

**1. Definition via CSS:**  Control.getUserAgentStylesheet():String

* Auslesen der -fx-skin CSS Property zur Referenzierung des Skins (in Control.loadSkinClass())
* **Nachteil:** Reflection

**2. Programmatische Instanziierung im Control:** createDefaultSkin():SkinBase

* **Vorteil:** Keine Reflection

## Entwicklung eigener Skins zu bestehenden Controls

* Java 7 vs. Java 8

[[Quelle: Styling UI Controls (OpenJFX)]](https://wiki.openjdk.java.net/display/OpenJFX/Styling+UI+Controls)

### A) Custom Control Properties wraps Sub Controls defined Skin vs. B) Bind Bidirectional Custom Control Properties in Skin

TODO: Draw Diagram

* A
  + (+) Performance (no further bindings necessary)
* B

(+) Reuse / rebind individual Custom Controls | Separation of Control and Skin

### Using Content Bias

TODO

## Zusatz: Testen von Controls

### Existierende Ansätze

#### JFXtras

* Welches Testframework?
  + TestFX 3.0.0 Beta 1
* Wie in Gradle integriert?

dependencies {

testCompile "junit:junit:${jfxtras\_junitVersion}","org.mockito:mockito-all:1.9.5"

**testCompile "org.loadui:testFx:3.0.0-beta1"**

asciidoclet 'org.asciidoctor:asciidoclet:0.+'

}

* Wie werden Test-Reports erstellt?

### Frameworks

#### TestFX

TODO

#### JemmyFX

TODO

### Integration mit Build Tools

### Gradle

#### Wie können die Tests in den Build integriert werden?

* Standardmäßig durch Einbinden in die Test-Compile-Phase (Task: **test** >Ausführen mit **gradle test**)

#### Wie können Test-Reports erstellt werden?

TODO

### Maven

## Zusatz: FXML Dynamic Root

### HowTo

TODO: Darstellung der Lösung mit Controller und Root (-Parent) getrennt durch Rückwärtsgenerierung UML Diagramm (Klassendiagramm mit FXML-Artefakten).

### Import in den SceneBuilder

Problem: Import der JAR, die eine CustomCompontent enthält, die mit FXML-Dynamic-Roots erstellt wurde, in den SceneBuilder ist nicht möglich.

Ursache: Das dynamische Laden von FXML-Dynamic-Roots wird durch den SceneBuilder nicht erkannt. So erstellte Custom Controls können nicht mit einer JAR ausgeliefert werden.

### Fazit

* (+) Kompaktheit / Prägnanz:
  + Kapseln des Ladevorgangs für die FXML mit dem Controller möglich
  + Controller und Root (-Node) können durch dieselbe Klasse repräsentiert werden (**ABER:** Geschäftslogik wird direkt in der View definiert > enge Kopplung zwischen Controller und View > vom Controller unabhängiger Austausch der View nicht möglich)
* (+) Deklarative Beschreibung von Custom Controls möglich
* (-) Mit FXML Dynamic Roots erstellte Custom Controls können nicht als JAR in den SceneBuilder importiert werden. Die Custom Controls werden nicht als solche erkannt.
  + Workaround: Editieren durch gewöhnliches Öffnen der FXML im SceneBuilder + Einbinden durch FXML-Includes

### Quellen

[[Dzone]](http://java.dzone.com/articles/creating-custom-javafx), [[Oracle1]](http://docs.oracle.com/javafx/scenebuilder/1/user_guide/library-panel.htm), [[Oracle2]](http://docs.oracle.com/javafx/2/fxml_get_started/custom_control.htm)

## Zusatz: New CSS API (JavaFX 8)

**Use Case:** Styling von Properties eines Custom Controls mit CSS (Java8)

TODO

[[Quelle: CSS API (OpenJFX)]](https://wiki.openjdk.java.net/display/OpenJFX/CSS+API+to+support+custom+UI+Controls)

## Tools

### Shemnon Gradle JavaFX Plugin

[[Quelle]](https://github.com/shemnon/javafx-gradle)

### Verwendung von Custom Controls im SceneBuilder

* Importieren von FXML
* Import von Klassenbibliotheken
  + [[Feature Request: Using Third Party Components on SceneBuilder]](https://javafx-jira.kenai.com/browse/DTL-5348)
  + Können FXML-Dynamic-Root-Custom Controls verwendet werden?
    - Nein, diese lassen sich nicht auflösen.
  + Muss eine Sample JavaFX Application existieren?
    - Nein, auch nicht-ausführbare JARs lassen sich importieren.
    - Ausschließliche Vorr. für den Import in den Scenebuilder ist die Einhaltung der gegebenen Restriktionen zur Architektur zu UI Controls (siehe oben).
  + Library Path für Custom Controls:

c:\Users\stefan.illgen\AppData\Roaming\Scene Builder\Library\

### UML Tools für Java 8

**Anf.:** Reverse Engineering von Java 8

* ArgoUML

# FAQ

## Wird die SkinBase.compute..() Methoden-Familie auch verwendet, wenn der entsprechende Wert fix definiert wird?

Nein, in diesem Fall wird der fixe Wert verwendet. Auch wenn die entsprechende Property gebunden wird, ist das der Fall.

## Was ist BehaviorSkinBase?

…

## Wofür wird das Interface Initializable verwendet?

Dieses Inteface gibt die ausschließlich die Methode intialize(..) vor. Diese Methode wird durch FXMLLoader.load() aufgerufen.

## Wo sollten Inner-Bindings definiert werden?

Eine Control kann Bindings auf anderen Controls kapseln (Inner-Bindings). Solche Bindings sollten

## Darf man ein Skin für NICHT-Controls definieren?

Nein, in diesem Fall muss die Layout-Berechnung in der NICHT-Control vorgenommen werden.

## Warum sind Builder in JavaFX 8.0 als Deprecated gekennzeichnet?

**Ursache:** Probleme mit binärer Kompatibilität > in JDK 6 und JDK 7 mussten Generics verwendet werden > Builder arbeiten nicht mehr korrekt für JDK8 („Es fallen zusätzliche Kosten im Bytecode an.“)

“We made a mistake . . . our implementation has some intractable problems with respect to binary compatibility . . . this was accomplished using generics, and as it turns out, depended on two bugs in JDK 6 and JDK 7 in order to work (although we didn't know it at the time). Those bugs were fixed in JDK 8, and as a result, the builders no longer work correctly for certain cases . . . My proposal after having weighed the options is to phase out the Builders by deprecating them in 8 and removing them from the class path in 9. I believe that FXML or Lambda's or alternative languages all provide other avenues for achieving the same goals as builders but without the additional cost in byte codes or classes.”

**Workaround:** Verwundung der Standard-API der Controls (Kein Fluent/Functional Style!)

**Lösung:** FXML OR Java 8 Lambda's OR andere Sprachen (Wie in Verbindung mit Custom Controls? FXML Dynamic Roots don’t work!!!)

[[Quelle: Oracle Forum]](https://community.oracle.com/thread/2544323?tstart=0), [[Quelle: Proposal 4 Deprecating Builders]](http://mail.openjdk.java.net/pipermail/openjfx-dev/2013-March/006725.html)

## Warum werden Control Disposals nicht durch JavaFX unterstützt?

**Use Case:** Unregister ChangeListeners defined as Lambdas

JavaFX bietet gegenwärtig ausschließlich einen entsprechenden Hook via Skin.dispose(), eine entsprechende Methode für Controls ist allerdings geplant. Für die Zukunft ist allerdings auch dies geplant. [[Quelle]](https://wiki.openjdk.java.net/display/OpenJFX/UI+Controls+Architecture)