GEBIT Solutions

Die Experten für Java, Anwendungsentwicklung und Requirements Engineering

Design von Business Anwendungen mit JavaFX
Java Forum Stuttgart - Vortrag F1

Tom Krauß
Partner und Mitglied der Geschäftsleitung
GEBIT Solutions GmbH

Agenda

- **Motivation**
  - Das Versprechen von JavaFX 2.x
- **Kernkonzepte von JavaFX für Business Anwendungen**
  - Trennung von UI und Business Logik
  - Data Binding
  - Weitere Mechanismen
- **Fazit**

JavaFX
Historie

Version 1.x

- Als Konkurrenz zu Flash und Silverlight entwickelt
  - Noch kein Business Fokus
- Verwendung eigener Skriptsprache Pflicht
- Add-on zum Java Development Kit

Version 2.x

- Nun auch Fokus auf Business Anwendungen
- Scriptsprache eliminiert
- Bestandteil von Java 7
- Jede Menge neue Controls für Business Anwendungen
- Scene Builder

- **Der Anspruch**

> "JavaFX 2 is the next step in the evolution of Java as a rich client platform. It is designed to provide a lightweight, hardware-accelerated Java UI platform for enterprise and business applications."

Merkmale von UI Frameworks für Business Anwendungen

Anwendungsrahmen

- Ablaufsteuerung
- Authentifizierung
- Commands and Actions
- Konfiguration
- Personalisierung

UI Mapping von Daten

- Architektur
  - MVP, MVC, ..
- Binding
- Konvertierung
- Validierung
- Autorisierung
- Printing

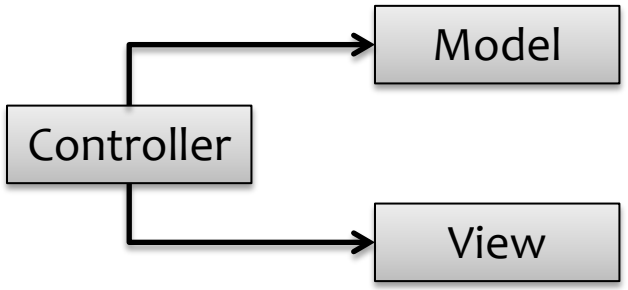
Präsentation

- Controls
- Effekte
- Skinning
- „Channels“
- Tools

Architektur: Das „Passive View“ Design Pattern
(Martin Fowler)

- **Definition**
  - *A screen and components with all application specific behavior extracted into a controller so that the widgets have their state controlled entirely by controller*

- **Eine Spielart des MVP (Model View Presenter) Patterns**
- **Keine Abhängigkeit zwischen Model und View**
  - (im Gegensatz zu MVC)

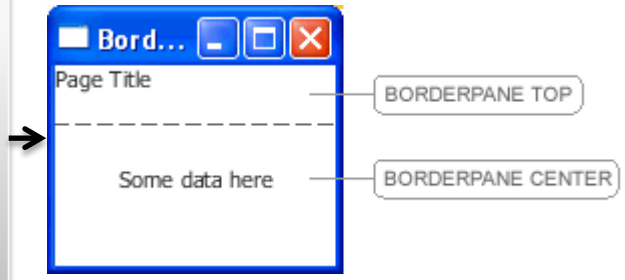


- **Verbessert die Testbarkeit von Anwendungen**
- **Views austauschbar**

Das „Passive View“ Design Pattern in JavaFX

- **FXML – ein basierter Objekterzeugungsmechanismus**
  - Definition des JavaFX UI Layouts – die Passive View
  - Skriptsprachen (JSR 223) und Java zur Definition der Dynamik
  - CSS für das Styling -> Skinning
- **Entkopplung vom Controller**
  - Dependency Injection zur Bindung von Methoden und Feldern

```
<BorderPane>
  <top>
    <Label text="Page Title"/>
  </top>
  <center>
    <Label fx:id="myLabel"
           text="Some data here"/>
  </center>
</BorderPane>
```



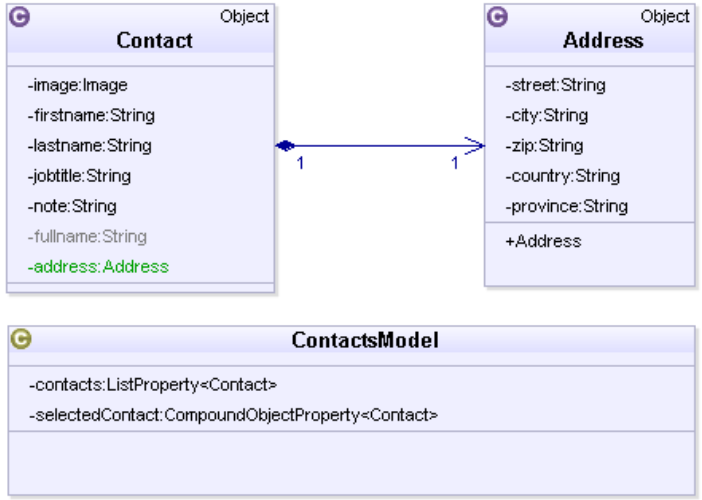

```
public class Controller
{
 @FXML
 private Label myLabel;
```

FXML und die Rollenverteilung im JavaFX Entwicklungsprozess

ContactsView.fxml	ContactsController.java	ContactsStyles.css
		
User Interface Designer	Applikationsprogrammierer Business Logic	Graphic Designer

Ein Beispiel
JavaFX und MVP für eine Kontaktlistenanwendung

Model



View

ContactsView.fxml

```
<BorderPane xmlns:fx="http://javafx.com/fxml"
            fx:controller="ContactsController">
<center>
    <VBox>
      <children>
        <AnchorPane>
         <TextField fx:id="firstname" />…
```

ContactsView.css

```
.root {
    -fx-font-size: 16pt;
    -fx-font-family: "Luicida";
}
```

Presenter

ContactsController.java

```
public class ContactsController {
   @FXML
   private TextField firstname;
   private ContactsModel model;

   @FXML
   protected void initialize() {
…
```

FXML – what's cool?

- **Strikte Trennung des Layouts vom Controller Code**
  - Alternative Views für einen Controller
- **Kann beliebige Objekte erzeugen – nicht nur UI Controls**
  - Beispiel 1: Definition eines Modells
  - Beispiel 2: Einbindung von Custom Renderern / Cell Editoren
- **Erweiterungsmöglichkeiten für Tools und Frameworks**
  - Definition beliebiger **`<properties>`** eines Controls
  - „Übergabe von Objekten" im „Namespace"
- **Kann zur Implementierung von „Custom Controls" verwendet werden**

FXML – Caveats

- **Entkopplung von View und Controller (Presenter) bzw. Modell nicht konsequent genug**
  - Controllercode nicht ablauffähig ohne „injizierte“ Controls
  - Viele Beispiele, bei denen Modelle in FXML definiert werden
- **Unzureichende Fehlerbehandlung, wenn Injection von UI Controls / Methoden in den Controller nicht erfolgreich**
- **Unzureichende Möglichkeiten des Bindens von FXML an Code**
  - Nicht alle Events per Binding an den Controller weiterleitbar
    Beispiel: Selection Change in Tabelle
  - Verwendung von „Expression Binding“
    gegen den Controller nicht möglich
  - Felder werden immer direkt und nicht über Setter „injiziert“

➢ Diese Punkte werden sicher nach und nach ausgeräumt

FXML – Caveats

Abhängigkeit der Controller von den Views

ContactController.java

```
public class Contact {
   @FXML
   private TextField firstname;
   @FXML
   private TextField lastname;
```

Schlecht: Abhängigkeit von konkreter Controlklasse

ContactController.java

```
public class Contact {
   @FXML
   private ITextEditor firstname;
   @FXML
   private ITextEditor lastname;
```

Besser wäre: Interfaces verwenden

- **Mocking**
- **Austauchbarkeit**

Properties und Binding

- **Erweitertes JavaBeans Modell**

Contact.java

```
public class Contact {
   private Property<String> firstname = new SimpleStringProperty();
   public Property<String> firstnameProperty() { return firstname; }
   public void setFirstname(String name) {
      firstname.set(name);
   }
   public String getFirstname() {
      return firstname.get();
   }
```

- **Erlaubt das „Binden“ an andere Objekte (z.B. Texteigenschaft eines Eingabefelds)**

ContactController.java

```
@FXML
private TextField firstname;
private Contact contact;

@FXML
public void initialize() {
   contact.firstnameProperty().
      bindBidirectional(firstname.text());
```

Binding und Konvertieren von Werten

- **Beispiel: ein Datum soll mit einem Textfeld editiert werden**
- **Lösung: Binden über einen StringConverter**

```
Bindings.bindBidirectional(birthday.textProperty(), contact.birthdayProperty(),
   new StringConverter<Date>() {
      @Override
      public Date fromString(String anInputString) {
            return myDateFormat.parse(anInputString);
      }
      @Override
      public String toString(Date aDate) {
            return myDateFormat.format(aDate);
      }
});
```

Properties und Binding – what‘s cool?

- **Extrem mächtiges Modell zur Kopplung „beliebiger“ Eigenschaften**
  - Beispiel: Rotation einer Tabelle – Winkel gekoppelt an Slider

```
<Slider fx:id="slider" min="0" max="100"/>
<TableView fx:id="contactBrowser" rotate="${slider.value}“/>
```

- **Wird konsequent im JavaFX Framework selbst genutzt**
  - Definition von Transitions und anderen Effekten, etc...
- **Mächtige High-Level ”Fluent API”**
  - Kombination von Werten einfach möglich

```
private StringExpression fullname = Bindings.concat(
   firstnameProperty(), “ “, lastnameProperty());
```

  - Flexible Datenkonvertierungen

Datenbindung und Verwendung von POJOS

- **Eigenschaften von POJOS können „out of the Box“ nicht gebunden werden**
- **Lösungsoption 1: für POJOS ein entsprechendes Properties Objekt definieren und Werte kopieren oder Pojos „wrappen“**
  - 👎 Resultiert in jeder Menge „Boilerplate“ Code
- **Lösungsoption 2: Value Objekte gleich mit JavaFX Properties definieren**
  - 👎 Abhängigkeit von JavaFX, Speicherbedarf, Serialisierung
- **Lösungsoption 3: Adapter verwenden**
  - JavaBeanProperty oder JFXTras BeanPathAdapter
  - 👍 Erlaubt das Binden von Eigenschaften von strukturierten POJOS
  - 👎 Löst auch nicht alle Probleme. Beispiel POJOS in einer Tabelle darstellen

Binding und POJOS

Lösungsoption 3 Verwendung des JFXtras BeanPathAdapter

ContactController.java

```
public class ContactController {
   @FXML
   private TextField street;
   @FXML
   private Labeled fullname;
   …

  @FXML
  public void initialize() {
    final BeanPathAdapter<Contact> adapter = new BeanPathAdapter<Contact>(…);
    adapter.bindBidirectional("fullname", fullname.textProperty());
    adapter.bindBidirectional("address.street", street.textProperty());
…
  }
```

Properties und Binding – Caveats

- **Validierung**
  - Noch nicht eingebaut in JavaFX 2.2
  - Proprietäre Lösungen existieren
  - Kann bei der Verwendung von „Binding“ schlecht von externen Frameworks übernommen werden
- **Binding ist „verführerisch“**
  - Beispiel: ein Eingabefeld zur Angabe eines Ehegatten ist nur bearbeitbar, wenn eine Checkbox „verheiratet“ angeklickt ist
    - Dies ist eine *Geschäftsregel*
    - Der enabled-Zustand des Eingabefelds sollte nicht vom „Checked“-Zustand einer Checkbox abhängen

Don’t!

```
<CheckBox fx:id="checkbox" text="Click me"/>
<TextField disable="${cb.selected}"/>
```

Controls

- **Viele „Standard“-Controls vorhanden**
  - Funktionalität und Umfang Standard vergleichbar zu Swing oder SWT
  - Gut: professionelle Chartcontrols + Webbrowser Control
- **Viele der „üblichen“ Business Controls fehlen:**
  - Maskierte Eingabe, Erweiterte Tabelle (Filtern etc...), Kalendercontrols, ...
  - Es entsteht bereits eine Reihe von 3rd Party-Lösungen: JFXtras, ControlsFX, JavaX UI Controls Sandbox
- **Definition eigener Controls schwierig**
  - Integrations-API in Scene Builder (vergleiche JavaBeans Modell) fehlt
  - Public Skin- und CSS-API (aktuell noch Oracle-interne Pakete) fehlt – angekündigt für JavaFX 8, aber noch nichts davon zu sehen ☹
- **Printing API**
  - Angekündigt und bereits enthalten in den JavaFX8 Snapshots

Was sonst noch so auffällt

- **Layout Management**
  - Nicht durch Delegation wie in anderen UI Frameworks gelöst, sondern durch spezielle Containerklassen
  - ☹ Unflexibler
  - ☹ Manager in JavaFX8 geändert?
- **Gewöhnungsbedürftige Definition von „typischen Delegates“ (Cell-Editoren, -Renderer, ...)**
  - Verwendung des abstrakten Interfaces „Callback“
  - Minimum 2 Klassen / Interfaces ableiten / implementieren
- **„Black-Box“ Sicht auf Controls**
  - Beispiel: keine Möglichkeit, den „sichtbaren Bereich“ einer Tabelle zu bestimmen
  - Immerhin: ab JavaFX 8 neue API zum Sichtbarmachen einer Zelle

Custom Renderer / Cell Editor:

```
firstnameColumn.setCellFactory(new
 Callback<TableColumn<Contact,String>,
  TableCell<Contact,String>>() {
      @Override
      public TableCell<Contact, String>
          call(TableColumn<Contact, String> aText) {
            return new TableCell<Contact, String>(){
               @Override
               protected void updateItem(
                 String text, boolean empty) {
                  super.updateItem(text, empty);
                  if (empty) {
                    return;
                  }
                  if ("Adele".equals(aText)) {
                     setTextFill(Color.RED);
                  } else {
                     setTextFill(Color.BLUEVIOLET);
                  }
…
```

JavaFX für Business Anwendungen
Zusammenfassende Beurteilung

Merkmal	Rating	Anmerkungen
Anwendungsrahmen	-	Noch nicht mal Message Boxen in der Core API enthalten
Ablaufsteuerung, und Rahmen- Konfiguration	-	Keine Lösung vorhanden, könnte aber durch 3rd Party beigesteuert werden
Actions	-	Eine Lösung findet sich in Add-On Library „ControlsFX“ – leider nicht mit Standardmenüs, ... verwendbar
Personalisierung	-	Keine Lösung vorhanden, könnte aber durch 3rd Party beigesteuert werden
Architektur MVP, ...	++	FXML
UI Mapping	+ (-)	Beans und Binding +, Konvertierung+, aber noch keine Validierung
Autorisierung	-	Keine Lösung vorhanden, könnte aber durch 3rd Party beigesteuert werden
Printing	- (+)	Geplant für JavaFX 8 (und in Snapshot Releases bereits vorhanden)
Controls	o	Umfangreiche Bibliotheken mit geliefert, weitere als Open Source verfügbar, Erweiterbarkeit noch eingeschränkt
Effekte	+++	Schon im Namen ;-). Umfangreiche API
Skinning	+++	Mächtig über CSS, allerdings kein Plattform Look&Feel, Neues Theme in JavaFX 8
Tools	++	Scenebuilder, ScenicView + weitere textuelle (NetBeans, IntelliJ) Allerdings keine Plugins für Scene Builder

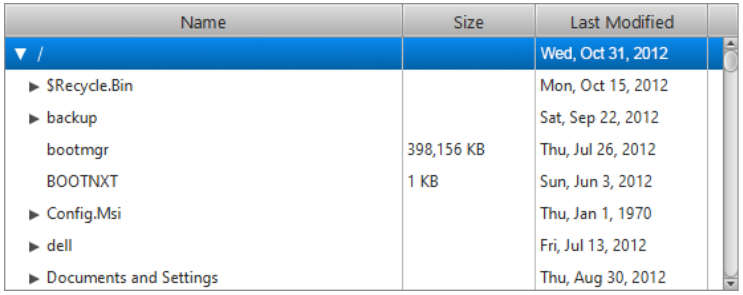
JavaFX 8
Was kommt und wo geht die Reise hin?

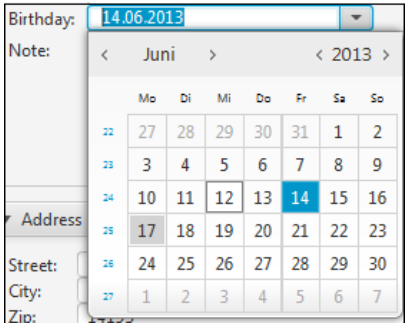
- **Printing**
- **Neue Business Controls (DatePicker, TreeTables, ...)**
- **Rich Text Support**
- **Support zur Definition eigener Custom Controls**
- **Unterstützung für Retina Displays**
- **Neues Theme „Modena“**
- **3D**
  - Moveable Camera
  - Lights & Materials (3D Attributes)
- **JavaFX im Standard-Classpath**

Diese Features sind bereits in aktuellen Builds enthalten

Leider noch nicht wirklich enthalten?

- **„Assistive Technologies“**
- **Abhilfe durch Access Bridge?**





Fazit

- **JavaFX 2.0 hat einen großen Schritt in Richtung „Framework für Business Anwendungen“ gemacht**
- **Anspruch, die neue UI Plattform für Multichannel Anwendungen zu sein, muss sich erst noch beweisen**
- **JavaFX 8 wird weitere Verbesserungen für Geschäftsanwendungen bringen**
  - Leider erst mit Java 8 verfügbar
  - In Zukunft sind Neuerungen dann leider immer an ein neues Java Release gekoppelt
- **JavaFX heute sicher schon als Plattform für Business Anwendungen einsetzbar**
- **Für den Einsatz im „großen“ Stil allerdings bleiben einige Fragen unbeantwortet**