Inhalt der Vorlesung

1. Klassen und Objekte
2. Vererbung
3. Enums, Wrapper und Autoboxing
4. Interfaces
5. Generics
6. Exceptions
7. Polymorphismus
8. Grafische Benutzeroberflächen mit JavaFX
9. Streams und Lambda Expressions
10. Leichtgewichtige Prozesse – Threads

Kapitel 8: Grafische Benutzeroberflächen mit JavaFX

Inhalt

Kapitel 8: Grafische Benutzeroberflächen mit JavaFX

Lernziele

[LZ 8.1] Die grundlegenden Konzepte (=Klassen) von JavaFX und deren Zusammenspiel kennen
[LZ 8.2] Die unterschiedlichen Möglichkeiten der Nutzung von Fenstern in JavaFX anwenden können
[LZ 8.3] Die von JavaFX vorgegebenen Controls und deren Möglichkeiten einsetzen können
[LZ 8.4] Die Panes von JavaFX kennen und zum Layout eines Fensters einsetzen können
[LZ 8.5] Das Layout eines Fensters in Form einer Layout-Hierarchie darstellen können

8. Grafische Benutzeroberflächen mit JavaFX

8.1 Überblick

Grafische Benutzungsoberflächen (Graphical User Interface, kurz GUI) werden heutzutage unter Einsatz sogenannter GUI-Bibliotheken bzw. GUI-Frameworks erstellt.

Für Java gebräuchlich sind JavaFX (Standard) und Swing (Vorgänger), Swing wird aber nicht mehr weiterentwickelt, daher besprechen wir im Folgenden JavaFX.

JavaFX bietet:

- Fenster als rechteckige Container für Bedienelemente
- Bedienelemente (Buttons, Textfelder, Auswahlmenüs, etc.) – auch Controls genannt
- Ereignisbehandlung (z.B. was passiert, wenn der Benutzer auf einen Button klickt?)
- hier u.a. nicht behandelt:
    - Layoutgestaltung über CSS (analog zu Webdesign)
    - Grafische Primitivoperationen (z.B. Linien und Punkte zeichnen, Farben setzen)
    - JavaFX FXML: XML-basierte Sprache zur Gestaltung von GUIs

Start einer JavaFX-Anwendung
Eine JavaFX-Anwendung wird nicht mit `main()` gestartet, sondern von einem sogenannten Launcher. Dies hat mit dem Lebenszyklus einer JavaFX-Anwendung zu tun, der nun erklärt wird. Eine JavaFx-Anwendung basiert auf der Basisklasse `Application`, von der typischerweise abgeleitet wird.

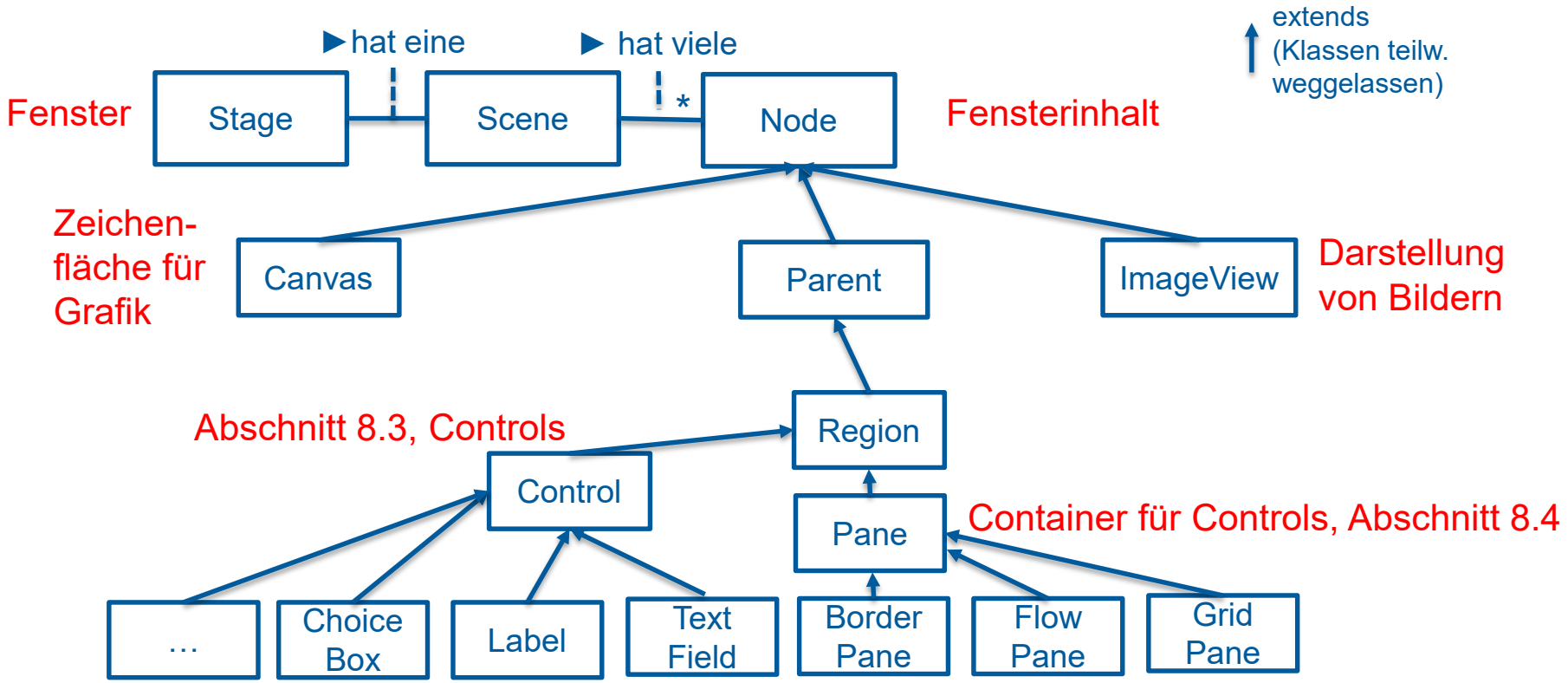
`Application` implementiert folgenden Lebenszyklus:
- beim Start der Anwendung wird eine Instanz von `Application` erzeugt, auf der dann die Methode `init` aufgerufen wird.
- `init` ist bereits in der Klasse `Application` leer definiert und kann, muss aber nicht überschrieben werden.
- Anschließend wird die Methode `start(Stage stage)` aufgerufen, der ein Fenster (Klasse `Stage`) übergeben wird, in dem der erste Scene-Graph (s.u.) dargestellt wird. Die Start-Methode sollte von jeder Application überschrieben werden. Hier können dann GUI-Controls erzeugt und in die Stage gesetzt werden.
- Wird die Anwendung mit `Platform.exit()` beendet, wird vor dem Beenden des Java-Prozesses die Methode `stop` aufgerufen, die bei Bedarf überschrieben werden kann, z.B. um offene Dateien zu schließen.

Eine JavaFX-Anwendung nutzt typischerweise folgende Objekte:

- Fenster werden in JavaFX durch Instanzen der Klasse `Stage` dargestellt (genauer siehe Abschnitt 8.2).
- Eine Stage enthält einen sog. *SceneGraph*, der ein Baum von `Nodes` ist. Jedes Node-Element ist typischerweise ein `Region-Objekt` oder ein `Control-Objekt` (es gibt noch andere Subklassen, auf die wir hier aus Zeitgründen nicht eingehen).
- Region-Objekte können `Panes` sein, die sich weiter in Border-, Flow- und GridPane aufgliedern. Diese werden für das Layout von Fenstern eingesetzt (s. Abschnitt 8.4).
- Control-Objekte sind die GUI-Controls, also die Bedienelemente innerhalb jedes Fensters. Diese werden in Abschnitt 8.3 erklärt.

Unter https://docs.oracle.com/javafx/2/api/ findet man die API-Dokumentation zu JavaFX.

Das nachfolgende Klassendiagramm zeigt die von Node abgeleiteten Klassen zur Verdeutlichung des zuvor Gesagten. Zur Vereinfachung wurden manche „Zwischenklassen“ nicht dargestellt.



Fenster sind die Grundelemente jeder GUI. In JavaFX werden Fenster durch Instanzen der Klasse Stage realisiert. Nachfolgend das klassische „Hello World!“-Programm mit JavaFX:

```
public class HelloWorldMitJavaFX extends Application {
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("Hello World Fenster");
        Pane pane = new FlowPane();
        pane.getChildren().add(new Label("Hello World!"));
        Scene scene = new Scene(pane, 300, 200);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    static public void main(String[] args) { launch(args); }
}
```

Die Anwendung wird durch den Befehl `launch` in main gestartet (s. Lebenszyklus), woraufhin eine HelloWordMitJavaFX-Instanz erzeugt wird, auf der `start` mit einem Fenster als Parameter aufgerufen wird. Der Titel des Fenster wird dann gesetzt und das Fenster erhält eine erste (und hier einzige) Szene mit einem Label-Element darin, welches den Text „Hello World!“ anzeigt. Am Ende wird das Fenster „primaryStage“ angezeigt. Standardmäßig werden die drei Systembuttons (Minimieren, Maximieren, Schließen) angezeigt. Der Schließen-Button beendet die Anwendung. Das Fenster ist in der Größe veränderbar (resizable).

Es können auch mehrere Fenster angezeigt werden. Hierfür sind lediglich mehrere Stage-Instanzen zu erzeugen. Das Beispiel weiter unten zeigt zwei Fenster an, wobei das zweite ein modales Fenster ist, d.h. es muss zuerst bedient werden. Erst wenn es geschlossen wurde, kann das erste Fenster genutzt werden.

```
public void start(Stage primaryStage) throws Exception {
    primaryStage.setTitle("Fenster 1 (amodal)");
    Pane pane = new FlowPane();
    pane.getChildren().add(new Label("Hello World!"));
    Scene scene = new Scene(pane, 300, 200);
    primaryStage.setScene(scene);
    primaryStage.show();
    Stage fensterZwei = new Stage();
    fensterZwei.setTitle("Fenster 2 (modal)");
    fensterZwei.setScene(new Scene(new Label("Schliesse mich!"), 300, 100));
    fensterZwei.initOwner(primaryStage);
    fensterZwei.initModality(Modality.WINDOW_MODAL); // Modales Fenster
    fensterZwei.setStyle("-fx-background-color: lightgrey");
    fensterZwei.show();
}
```

Erläuterung des Beispiels:

- Die Methode setTitle legt den Titel eines Fensters fest.
- Eine FlowPane ist ein rechteckiger Container, dessen Elemente (Children) von links nach rechts angeordnet werden (Flow-Layout, s. Abschnitt 8.4).
- Ein Label-Objekt enthält Text, welcher an beliebiger Stelle angeordnet werden kann.
- Es wird eine erste Scene mit der zuvor erzeugte FlowPane als Wurzelelement erzeugt
- mit dem Befehl show wird das erste Fenster („primaryStage“) sichtbar gemacht
- Das zweite Fenster wird analog zum ersten erzeugt und mit Inhalt gefüllt, statt einer FlowPane wird diesmal direkt ein Label als Wurzelelement der Scene gesetzt.
- Durch initOwner wird eine Eltern-Kind-Beziehung zwischen beiden Fenstern etabliert. InitModality legt fest, dass das zweite Fenster modal ist. Dies ist insbesondere bei klassischen Eingabedialogen oder bei Einstellungsdialogen (vgl. Eclipse „Preferences“) wünschenswert.
- Die Methode setStyle(String) setzt einer Node einen CSS-Style. CSS ist die aus der Webentwicklung bekannte Layout-Sprache.

Die Fenster einer grafischen Benutzeroberfläche stellen typischerweise Informationen dar und bieten in der Regel Möglichkeiten, die angezeigten Informationen zu verändern. Für diese Zwecke bietet JavaFX eine Menge von vordefinierten Bedienelementen (Controls) an. Wie bereits erwähnt (s. Klassenmodell) werden diese alle von der Klasse Control abgeleitet, die wiederum von Parent erbt. Controls werden angezeigt, in dem man sie der Kinderliste eines geeigneten Parent-Objekts hinzufügt. Jedes Parent-Objekt hat eine Liste von Kind-Objekten (*children*). Geeignete Parent-Objekte sind alle Arten von Container, wie z.B. die FlowPane aus einem vorherigen Beispiel.

```
Pane container = new FlowPane();                    // Container erzeugen
container.getChildren().add(new Label("Ein Text")); // Label hinzufügen
```

Folgende JavaFX-Controls werden besprochen:

- Button: Aktionsschaltfläche zum Anklicken
- CheckBox: Auswahlkasten (gecheckt, nicht gecheckt, undef.)
- ChoiceBox und ComboBox: Auswahlliste für wenige bzw. viele Elemente
- Label: Textfeld zur Anzeige von Text
- ListView: Liste von Objekten mit Einfach-/Mehrfachselektion
- PasswordField: Passworteingabefeld mit verdeckter Eingabe
- RadioButton: Einzelauswahl aus einer Gruppe von Optionen
- TableView, TableColumn, TableCell: Tabelle mit Spalten (TableColumn) und Zellen (T-Cell)
- TextField und TextArea: Einzeilige bzw. mehrzeilige Texteingabe
- ToogleButton: Button mit den Werten selektiert/nicht selektiert

Folgendes Beispiel zeigt den Einsatz einiger der zu besprechenden Controls:

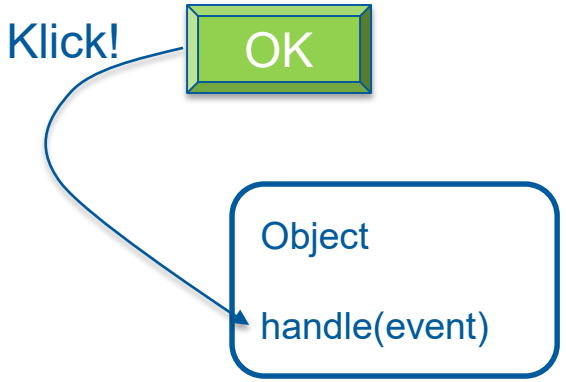
```
public class Beispiel1 extends Application {
    public void start(Stage primaryStage) {
        TextField nameTextField = new TextField();
        Button okButton = new Button("OK");
        primaryStage.setTitle("Namenseingabe");
        Pane pane = new FlowPane();
        pane.getChildren().add(new Label("Name: "));
        pane.getChildren().add(nameTextField);
        pane.getChildren().add(okButton);
        Scene scene = new Scene(pane, 300, 50);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    static public void main(String[] args) { launch(args); }
}
```

Zuerst werden die drei Controls (Textfeld, Button, Label) erzeugt, dann per add-Befehl der FlowPane als Kinder hinzugefügt. Wird der Button gedrückt, passiert – nichts! Das liegt daran, dass wir keine Ereignisbehandlung definiert haben.

Ereignisbehandlung
Im Folgenden soll das Konzept der Ereignisbehandlung anhand des letzten Beispiels erklärt werden. Jedes Control kann auf eine Benutzerinteraktion hin bestimmte Ereignisse auslösen, ein Button löst bspw. ein `ActionEvent` aus, wenn er gedrückt wird.
Um eine Reaktion auf den Button-Klick zu definieren, muss eine sog. Handler-Methode beim Button-Objekt selbst registriert werden. Die Methode wird dann beim Button-Klick durch das JavaFX-Framework aufgerufen. Hier lässt sich auch der Unterschied zwischen Framework und Bibliothek erklären: das Framework ruft den „user code“ auf, die Bibliothek wird vom „user code“ aufgerufen (z.B. double val = Math.sin(3.2);)

Für das letzte Beispiel könnte man bspw. festlegen, dass bei jedem Button-Klick der in das Textfeld eingegebene Name auf die Console ausgegeben werden soll:

```
okButton.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent event) {
        System.out.println("Name: " + nameTextField.getText());
    }
});
```



Der Methode setOnAction wird eine Instanz einer sogenannten anonymen Klasse übergeben. Von der oben definierten anonymen Klasse ist nur bekannt, dass sie das EventHandler-Interface mit dem Parameter ActionEvent implementiert. Die handle-Methode ist der eigentlich zu spezifizierende Handler für das Ereignis. Man könnte auch Instanzen anderer Klassen übergeben und diese speziell für diesen Zweck definieren. Obige Methode ist jedoch zu bevorzugen, da sie weniger Code erfordert. Mit Lambdas (s. Kapitel 10) lassen sich Event-Handler noch kürzer formulieren.

Der zuletzt erwähnte Weg über eine eigene Klasse sähe etwa wie folgt aus:

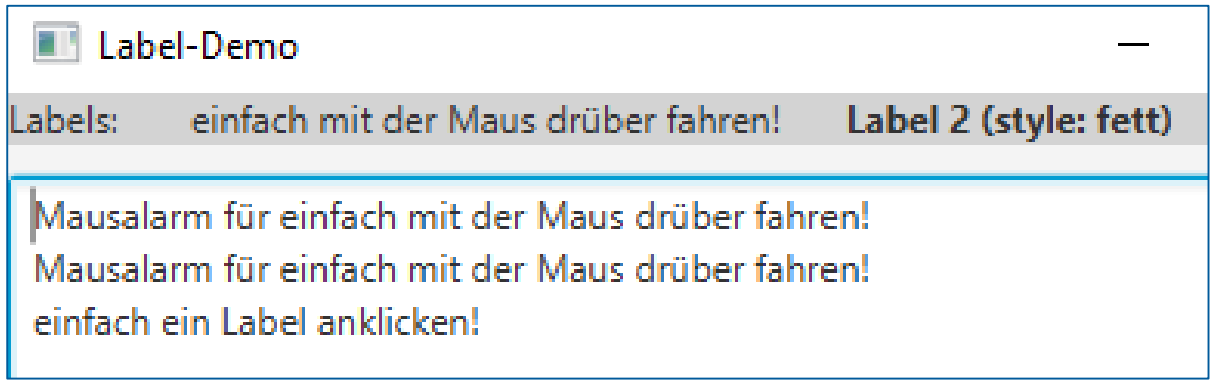
```
// Handlerklasse mit handle-Methode für Ereignisbehandlung
class MyHandler implements EventHandler<ActionEvent> {
     private TextField nameTextField;
     public MyHandler(TextField nameTextField) {
          this.nameTextField = nameTextField;
     }
     public void handle(ActionEvent event) {
          System.out.println("Name: " + nameTextField.getText());
     }
}
...
// Erzeugen und Setzen des Handlers
okButton.setOnAction(new MyHandler(nameTextField));
```

Die Handlerklasse benötigt eine Referenz auf das Textfeld für den Namen, um den eingegebenen Namen ausgeben zu können. Man sieht auf einen Blick, dass diese Lösung sehr umständlich ist und mehr Code erfordert. Mit einem Lambda-Ausdruck (vgl. Kapitel 9) ergäbe sich:

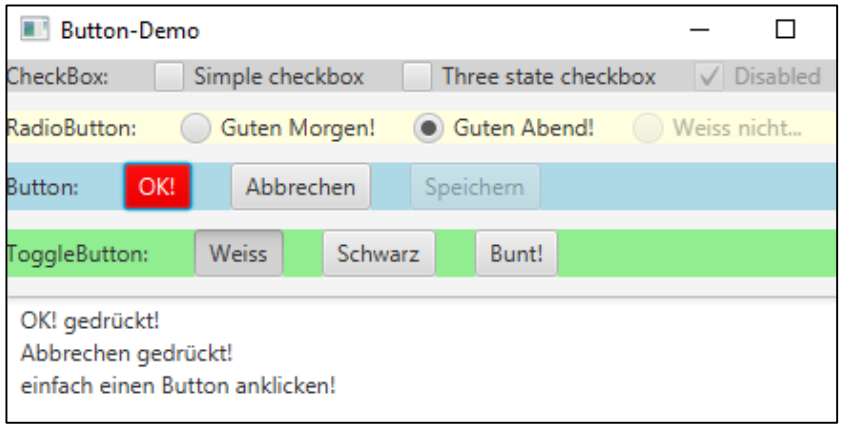
```
okButton.setOnAction(e -> System.out.println("Name:"+ nameTextField.getText()));
```

Da die unterschiedlichen Controls unterschiedliche Events auslösen, werden die Events und die entsprechenden Eventhandler zusammen mit den Controls erklärt.

Labels
Ein Label dient zur Darstellung von Text, der positioniert und gestylt werden kann.
Eigenschaften:
- String-Attribut text mit Getter/Setter
- Konstruktor mit String-Parameter zur Initialisierung des Texts
- Ereignisbehandlung (selten):
    - setOnMouseEnter(EventHandler<MouseEvent>): wird aufgerufen, wenn der Mauszeiger über das Label bewegt wird



Klasse Labels, s. Kursraum



Klasse Buttons, s. Kursraum

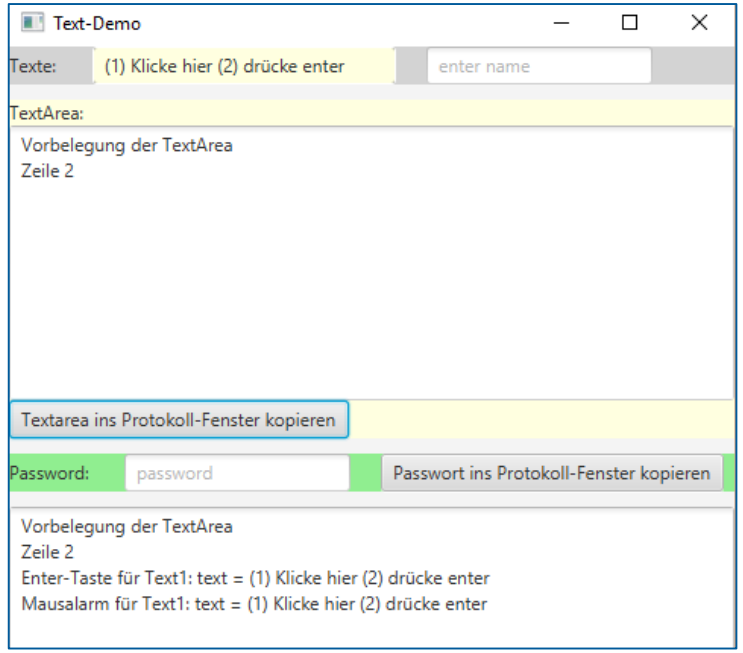
Button, CheckBox, RadioButton, ToggleButton

Alle Buttons haben bestimmte Eigenschaften, u.a.

- Attribut „text“: Beschriftung des Buttons
  - Zugriff über Getter/Setter
- Konstruktor: Übergabe von Button-Beschriftung als Parameter
- Ereignisbehandlung
  - ausgelöstes Ereignis bei Klick: ActionEvent
  - Setzen des Event-Handlers: setOnAction(EventHandler<ActionEvent>)
- 1 aus N-Auswahllogik durch Bildung einer ToggleGroup: Neben dem Hinzufügen zu einem Container zur Anzeige müssen Radio- bzw. ToggleButtons zu einer ToggleGroup zusammengeführt werden. Hierzu gibt es die Methode setToggleGroup(ToggleGroup) an den Buttons
- Setzen der Style-Attribute wie bei CSS (s. Webseiten) mit setStyle(<Style-String>), Beispiel siehe OK!-Button (gilt für alle Controls und Container, s. FlowPanes)
- Selected-Eigenschaft bei CheckBoxen, RadioButtons und ToggleButtons:
  Getter: boolean isSelected();
  Setter: void setSelected(boolean)
- Checkboxen können drei Zustände haben:
  „checked“, „unchecked“ (Standard) und als Option „undefined“: Diese muss explizit erlaubt und abgefragt werden, s. Button-Demo.

TextField, TextArea, PasswordField
Alle drei Klassen stellen Text dar, TextArea kann mehrzeiligen Text mit automatischen Scrollbars darstellen.

Eigenschaften:
- String-Attribut text mit Getter/Setter
- Konstruktor mit String-Parameter zur Initialisierung des Texts (nicht bei PasswordField)
- Ereignisbehandlung
  - setOnMouseClicked(EventHandler<MouseEvent>): Mausklick in Textfeld
  - setOnAction(EventHandler<ActionEvent>): Enter-Taste in Textfeld gedrückt



Klasse Texts, s. Kursraum

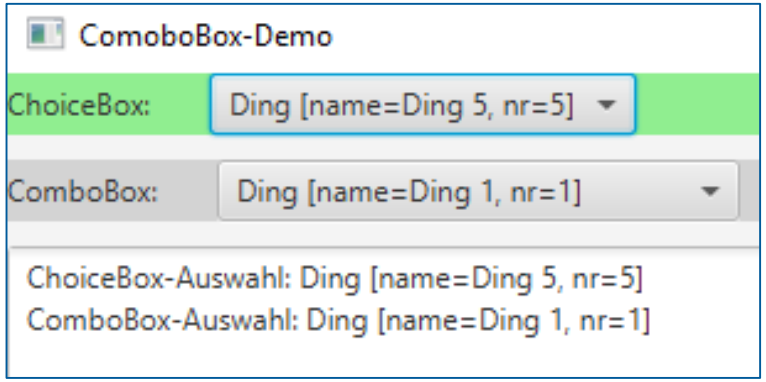
ChoiceBox, ComboBox

ComboBoxen, auch Dropdowns genannt, dienen der Auswahl eines Eintrags aus einer Liste. Für den Auswahlprozess wird die Liste ausgeklappt, nach der Auswahl klappt sie wieder zusammen und zeigt den gewählten Eintrag. Die Klasse ChoiceBox ist für die Auswahl aus kleinen Listen, ComboBox für die Auswahl aus großen Listen geeignet.

Eigenschaften:

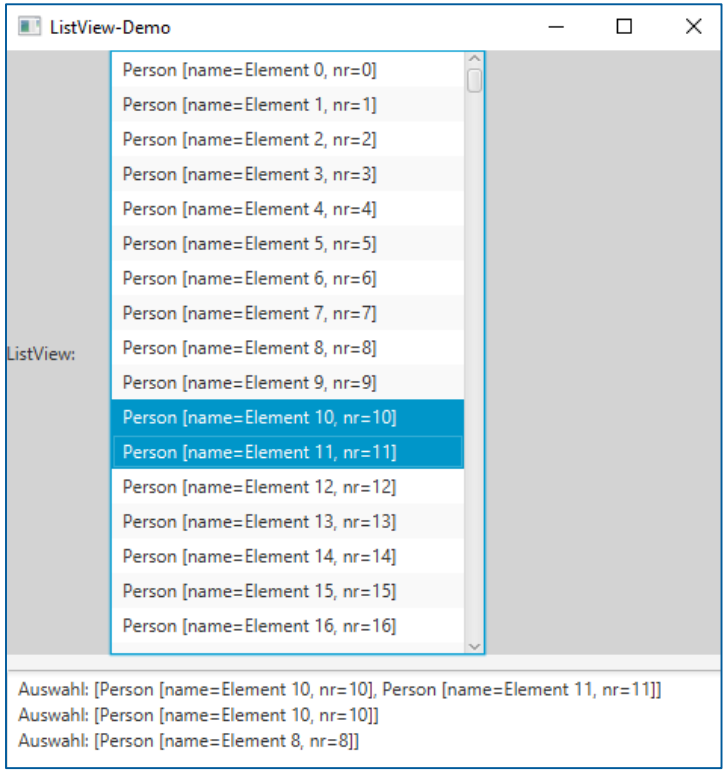
- Attribut ObservableList<Listenelement-Typ> items mit Getter/Setter; ein Beispiel für die Erzeugung und Befüllung einer ObservableList findet sich in der Klasse ComboBoxes.
- Konstruktor mit optionalem Parameter für „items“
- Ereignisbehandlung:
    - setOnAction(EventHandler<ActionEvent>): Eintrag ausgewählt
- Zugriff auf selektiertes Element:

```
ComboBox<String> cb = new ComboBox<>(observableList);
String selectedItem = cb.getSelectionModel().getSelectedItem();
```



Klasse ComboBoxes, s. Kursraum

Klasse Listen, s. Kursraum



Listen mit ListView<T>

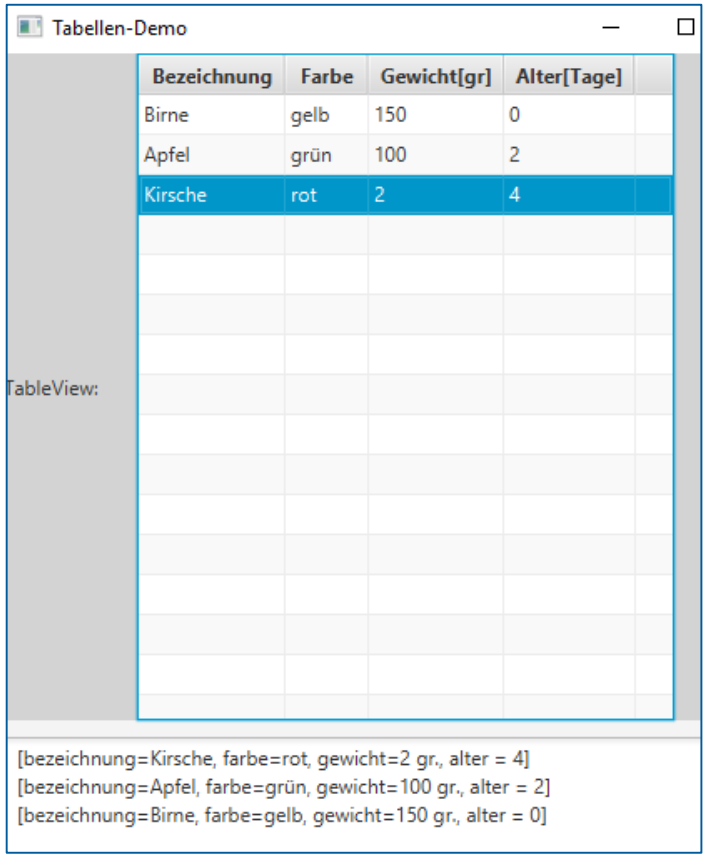
Eine ListView stellt eine ObservableList von Objekten eines bestimmten Typs T dar und ermöglicht die Auswahl eines oder mehrerer Objekte.

Eigenschaften:

- ObservableList<T> items: Attribut für Listenelemente mit Getter/Setter
- Konstruktor mit optionalem Parameter für „items“
- Mehrfach-Selektion setzen (Default ist Einfach-Selektion):

```
<listView>.getSelectionModel().
 setSelectionMode(SelectionMode.MULTIPLE);
```

- Ereignisbehandlung:
  `<listView>.setOnMouseClicked(EventHandler<MouseEvent>)`: falls Eintrag selektiert
- Lesen der selektieren Einträge:
  `<listView>.getSelectionModel().getSelectedItems()`



Klasse Tabellen, s. Kursraum

Tabellen
Analog zu ListView stellt eine TableView eine ObservableList von einem bestimmten Typ dar, wobei jedes Objekt in einer Zeile dargestellt wird.

Eigenschaften:
- ObservableList<T> items: Attribut für Listenelemente mit Getter/Setter
- Konstruktor mit optionalem Parameter für „items“
- Mehrfach-Selektion setzen (Default ist Einfach-Selektion):
  `<tableView>.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);`
- Ereignisbehandlung:
  `<tableView>.setOnMouseClicked(EventHandler<MouseEvent>)`: falls Eintrag selektiert
- Lesen der selektieren Einträge:
  `<tableView>.getSelectionModel().getSelectedItems()`

Tabellen (Fortsetzung)

Vorbereitung der Tabellenspalten

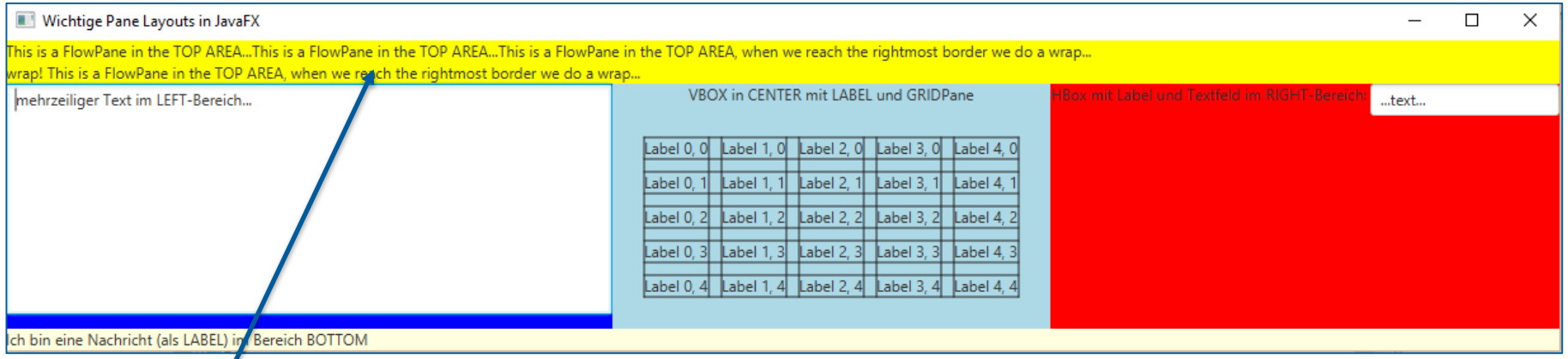
Bevor die Daten sichtbar werden können, muss für jede Spalte eine `TableColumn`-Instanz passend initialisiert werden. Hierzu wird der Spaltentitel gesetzt und es wird eine `CellValueFactory` gesetzt. Im Beispiel werden hierfür Instanzen der Klasse `PropertyValueFactory` genutzt, die den entsprechenden Attributwert anhand des Attribut-Namens (z.B. „bezeichnung“) per Java-Reflection aus dem Zeilen-Objekt durch Getter-Aufruf extrahiert (z.B. `getBezeichnung`).

Editierbare Tabellen

Möchte man die Zellen einer Tabelle editieren können, so muss zunächst `setEditable(true)` aufgerufen werden. Anschließend muss für jede Spalte, deren Zellen editiert werden sollen, eine `CellFactory` gesetzt werden, die sich um die Konvertierung der Zellen-Daten zwischen dem Objekt-Attribut (z.B. `bezeichnung`) und der Textdarstellung in der Zelle kümmert (Spalte „Bezeichnung“). Hierfür stellt sie einen Konverter zur Verfügung, der das Interface `StringConverter` implementiert. Im Beispiel wird ein Standard-Konverter für String-Attribute und ein eigener für die `int`-Attribute (`gewichtInGramm`, `alter`) genutzt.

Die explizite Positionierung der Controls eines Fensters anhand konkreter Koordinaten wird in der Praxis selten praktiziert, etwa weil das Fenster in der Größe veränderbar sein soll, Controls ein- bzw. ausgeblendet werden oder weil sich der verfügbare Platz (*screen resolution*) je nach Zielplattform ändert. Für all diese Fälle bietet JavaFX Pane-Klassen mit vorgegebenen Layout-Strategien an, von denen nun fünf Wichtige besprochen werden sollen:

- FlowPane: Anordnung der Controls von links nach rechts mit Umbruch an der rechten Container-Grenze
- BorderPane: Aufteilung des Container-Rechtecks in fünf Bereiche: Top, Bottom, Right, Left, Center, jeder Bereich kann ein Parent-Objekt (s. Vererbungshierarchie) enthalten
- GridPane: Matrix-artige Struktur mit beliebiger Anzahl Zeilen und Spalten (hängt von Befüllung ab!); jede Zelle der Matrix kann ein Parent-Objekt enthalten
- HBox: alle Controls werden in einer Zeile dargestellt
- VBox: alle Controls werden in einer Spalte dargestellt
- weitere (hier nicht besprochen): Anchor-, Tile-, Stackpane

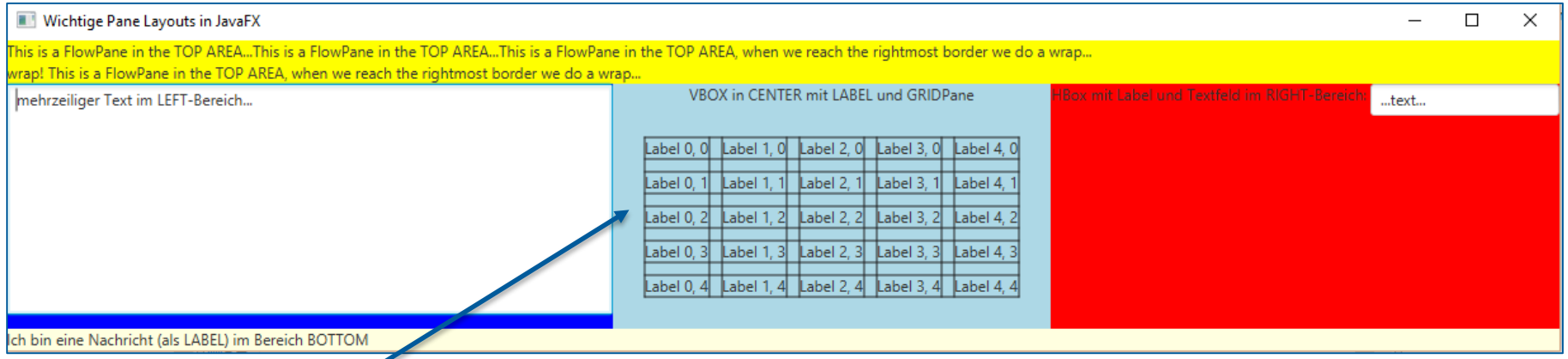


Klasse LayoutDemo, s. Kursraum

FlowPane

Eine FlowPane ordnet die hinzugefügten Elemente von links nach rechts an. Wird die rechte Pane-Grenze erreicht, so wird mit dem nächsten Element eine neue Zeile begonnen.

Vorgehensweise zum Hinzufügen von Elementen zu einer FlowPane:
Einzufügende Elemente (Pane oder Control) werden mithilfe der Operation add eingefügt:
<flowPane>.getChildren().add(<Parent>)



Klasse LayoutDemo, s. Kursraum

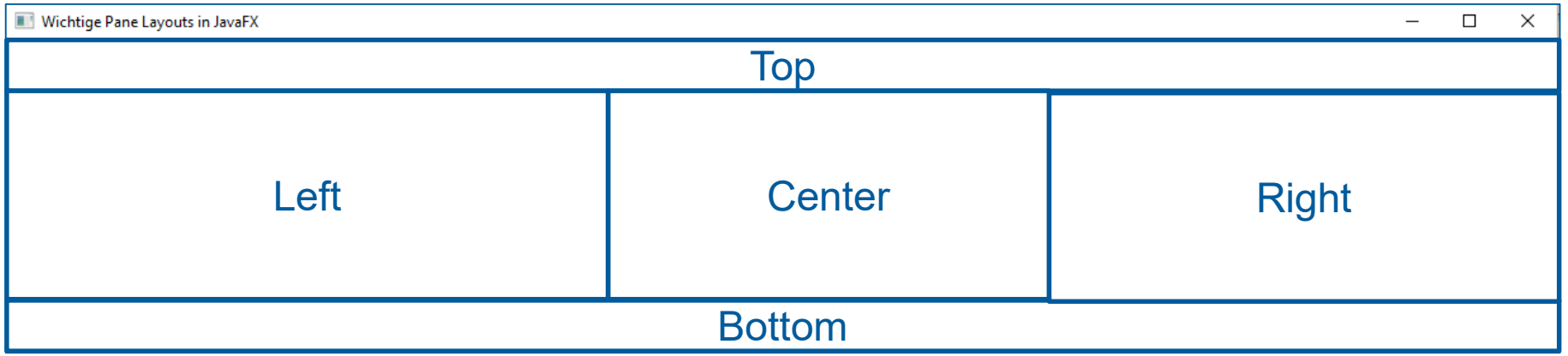
GridPane

Eine GridPane stellt eine Matrix-artige Struktur mit beliebiger Anzahl Zeilen und Spalten (hängt von Befüllung ab!) dar. Jede Zelle der Matrix kann ein Parent-Objekt enthalten. Die maximalen Dimensionen ergeben sich aus den maximalen in den add-Operationen verwendeten Zeilen- bzw. Spalten-Indizes.

Vorgehensweise zum Hinzufügen von Elementen zu einer GridPane:
Einzufügende Objekte (Pane oder Control) werden mithilfe der Operation

```
<gridPane>.add(parent, spalte, zeile)
```

eingefügt.

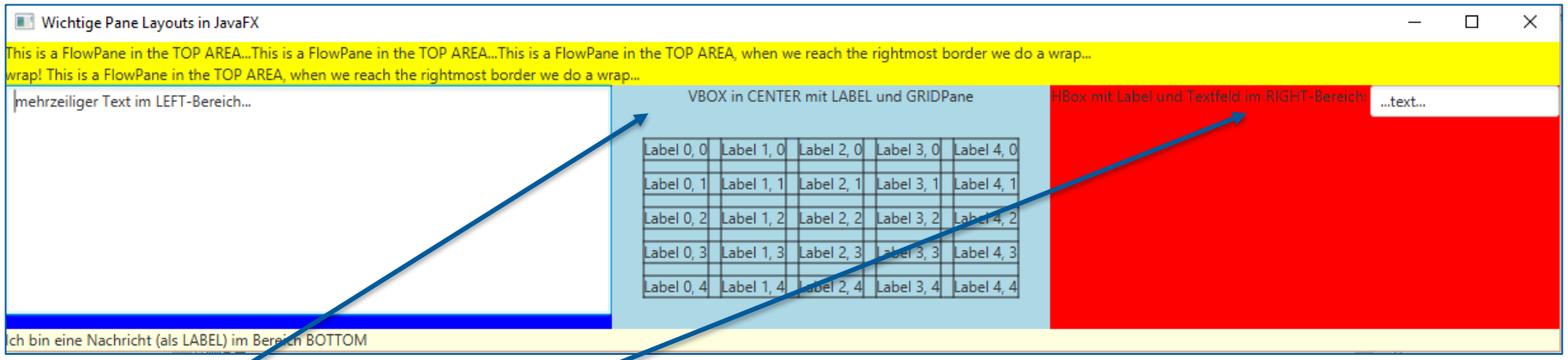


Klasse LayoutDemo, s. Kursraum

BorderPane
Eine BorderPane unterteilt das Container-Rechteck in die oben dargestellten fünf Bereich Top, Bottom, Right, Left und Center.

Vorgehensweise zum Hinzufügen von Elementen zu einer BorderPane:
Man fügt Panes oder Controls in die Bereiche ein, indem Setter-Operationen genutzt werden, etwa
`<borderPane>.setCenter(element);`
Für jeden Bereich existiert ein entsprechender Setter.

Beachte: Mehrere Controls (z.B. zwei Buttons) müssen in eine Pane gepackt werden, welche dann in die BorderPane an die entsprechende Position gesetzt wird.



Klasse LayoutDemo, s. Kursraum

VBox und HBox

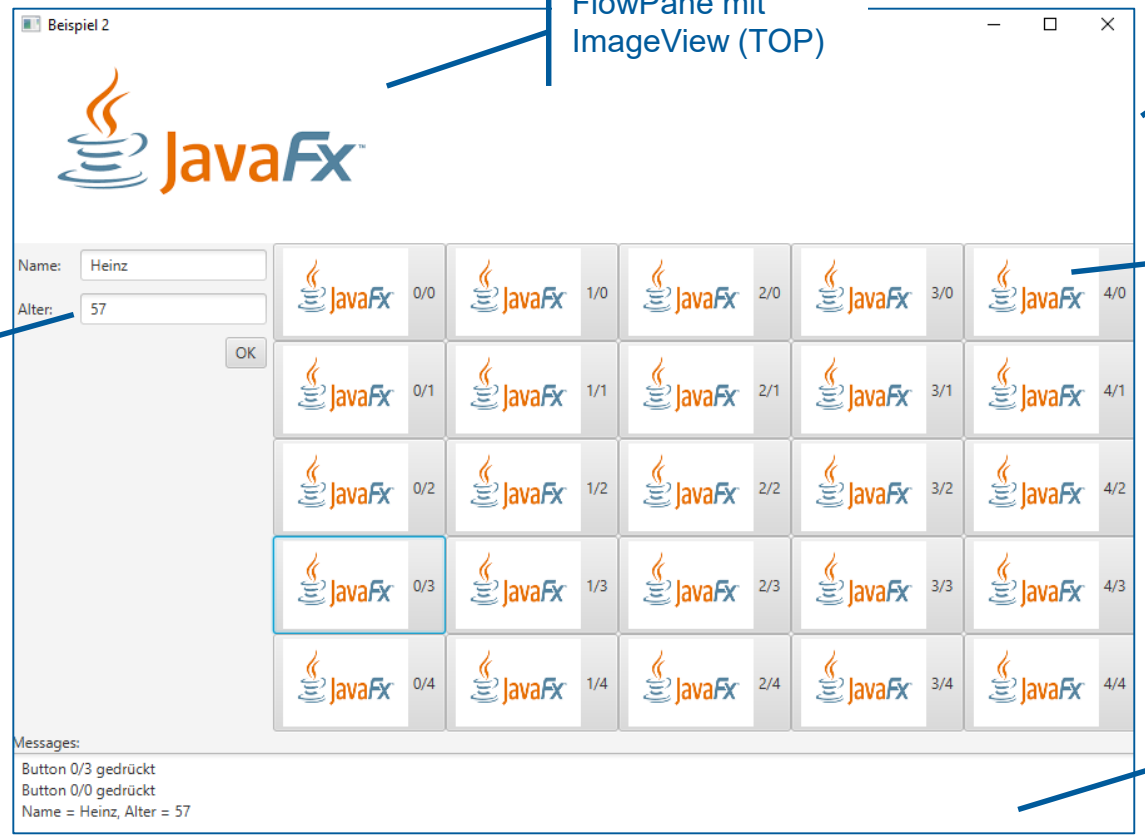
Eine `HBox` teilt den Container der `Pane` in eine Zeile ein, deren Breite von den von links nach rechts angeordneten Elementen sowie den gesetzten Zwischenräumen (mit `setHGap(double)`) abhängig ist. Analog teilt eine `VBox` den Pane-Container in eine Spalte ein, deren Höhe von den von oben nach unten angeordneten Elementen sowie den gesetzten Zwischenräumen (mit `setVGap(double)`) abhängig ist.

Vorgehensweise zum Hinzufügen von Elementen:
Einzufügende Elemente (Pane oder Control) werden mithilfe der Operation `add` eingefügt:

```
<hbox/vbox>.getChildren().add(<Parent>)
```

Beispiel für Layout- und Control-Verwendung

Die im Kursraum zu findende Klasse „Beispiel2“ zeigt die Verwendung einiger der erklärten Panes und Controls. Unten ist das Groblayout gezeigt, Erläuterungen s. Quelltext…



FlowPane mit ImageView (TOP)

Border Pane

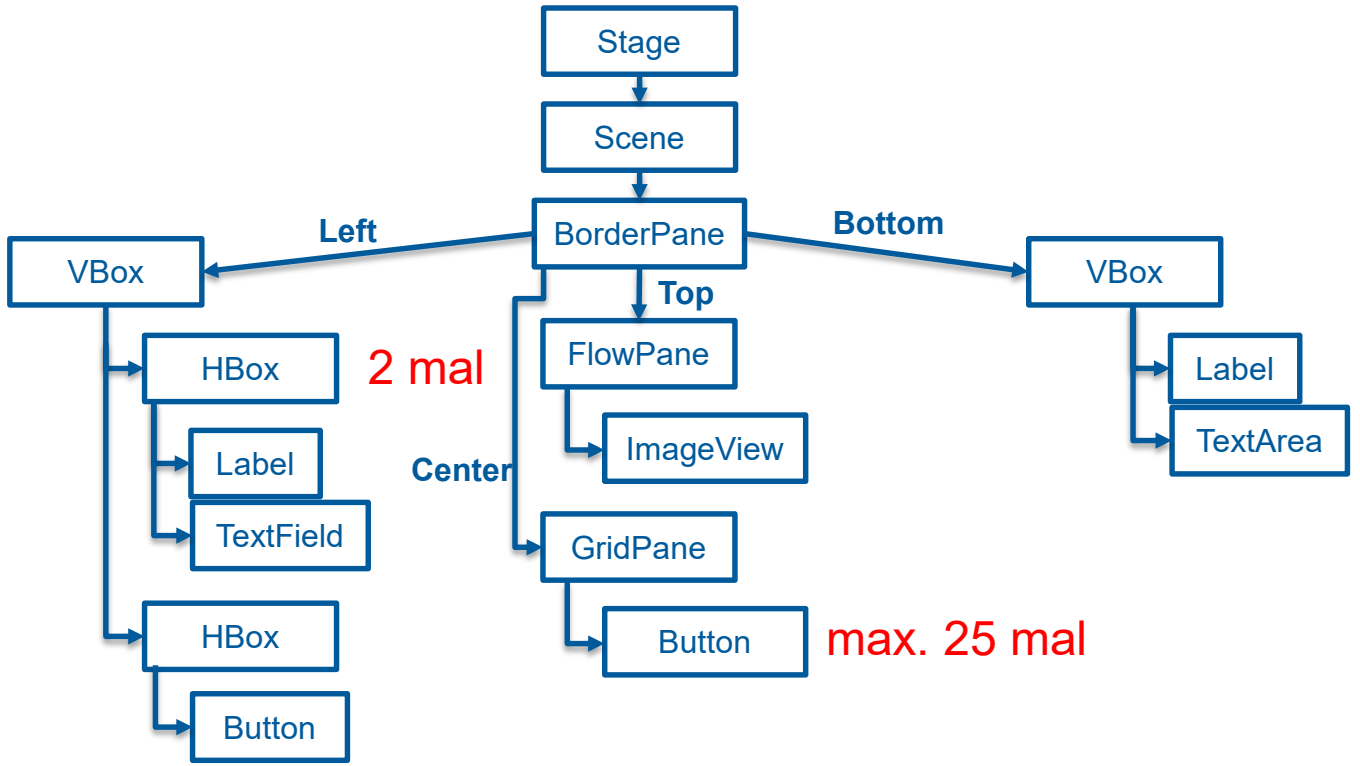
Hbox‘en als Zeilen in VBox
Je HBox: Label + TextField oder Button (LEFT)

Buttons mit Image View in GridPane (CENTER)

Text Area (BOTTOM)

Layout-Hierarchie des Beispiels

Zur einfacheren Nachvollziehbarkeit des Layouts kann eine sog. Layout-Hierarchie als Baumstruktur herangezogen werden. Sie stellt die beteiligten Objekte mit ihren Typen sowie etwaige Layout-Informationen dar. Man gewinnt Sie „bottom-up“ aus dem Quellcode oder im Rahmen eines Design-Prozesses „top-down“, typischerweise aus Papier-Skizzen.



8.5 Quellen

- JavaFX 8 – Grundlage und fortgeschrittene Techniken; A. Epple, dpunkt.verlag, 1. Auflage 2015
- online: https://docs.oracle.com/javafx/2/api/