Java GUI Programmierung

Michaela Pum

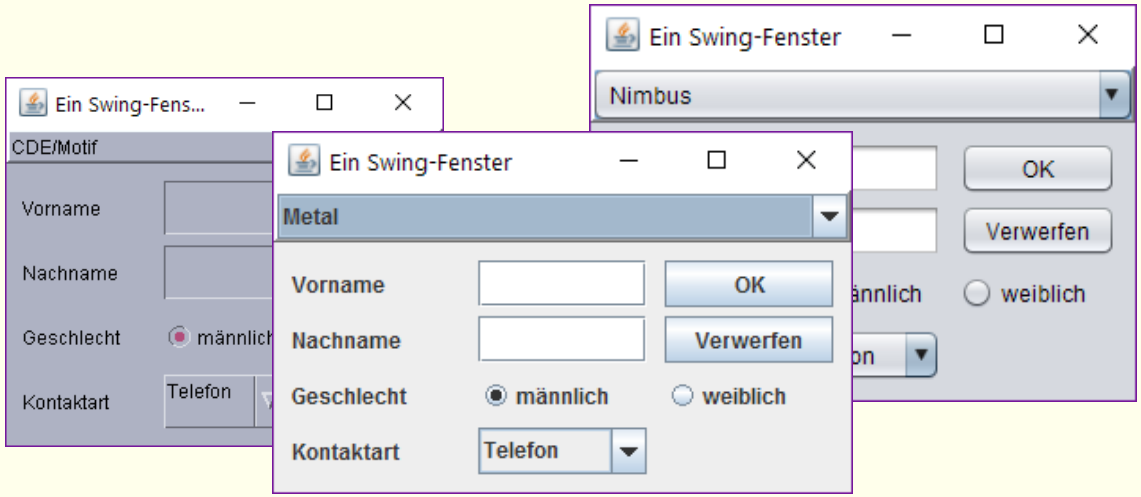
GUI Frameworks

- **AWT (Active Windowing Toolkit)**
  - ab Java 1.0
  - verwendet die Komponenten des Betriebssystems
    - unterschiedliches Aussehen und Verhalten
    - nur wenige Grundkomponenten
  - definiert Eventsystem und Layout für AWT und Swing Applikationen
- **Swing**
  - ab Java 1.1
  - Komponenten werden gezeichnet
    - gleiches Aussehen und Verhalten auf allen Plattformen
    - beliebige Komponenten können designt werden

GUI Frameworks

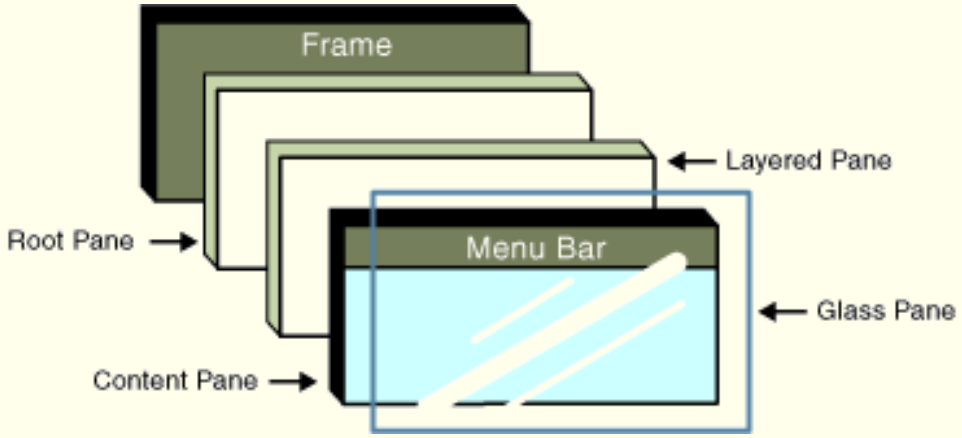
- **Java FX**
  - Neues Framework für Desktopanwendungen und Browser-Applets
  - verwendet je nach Plattform eine passende Rendering-Engine
    - DirectX unter Windows
    - OpenGL auf Linux oder iOS
  - verwendet wenn möglich Hardware-Beschleunigung
  - Entwicklung
    - Seit 2008 parallel zu Swing entwickelt
    - In Java 7 und 8 Teil von Java SE
    - Seit Java 9
      - unter dem Namen org.openjfx als open source Projekt verfügbar
      - aber nicht mehr Teil der Java SE

GUI Programmierung mit Swing

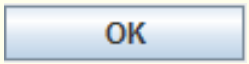


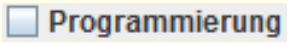
Swing Komponenten

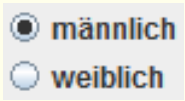
- **JFrame**
  - Basisklasse für Toplevel-Fenster
    - Titelleiste, Rahmen
    - optional weitere Komponenten, Menüleiste
  - ist in mehrere Ausschnitte ("pane") gegliedert:
    - **Content Pane**: enthält die Komponenten
    - **Layered Pane**: enthält die Content Pane und das Menü
    - **Glass Pane**: zum darüber Zeichnen, defaultmäßig unsichtbar
    - **Root-Pane**: enthält alle anderen



Swing Komponenten

JButton 

JCheckBox 

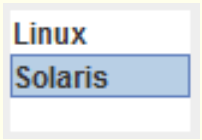
JRadioButton 

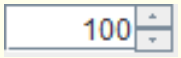
JTextField 

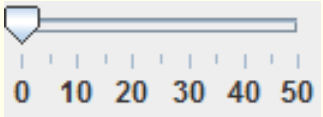
JPasswordField 

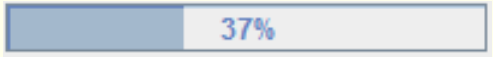
JTextArea 

JComboBox 

JList 

JSpinner 

JSlider 

JProgressBar 

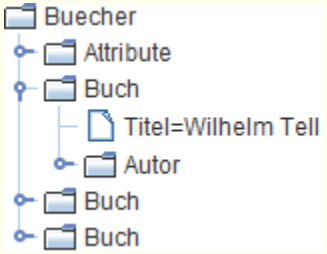
JLabel Zuname

Swing Komponenten

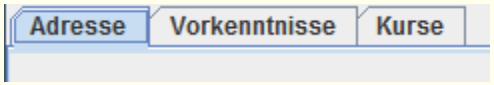
JTable

Bezeichnung	Preis	Status	Lieferbar
Schraube klein	0,0300	Lieferbar	☑
Schraube groß	0,0600	Lieferbar	☑
Schraube mittel	0,0400	AusVertrieb	☐

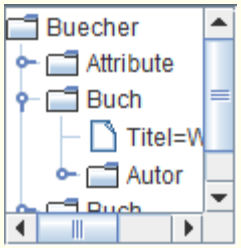
JTree



JTabbedPane



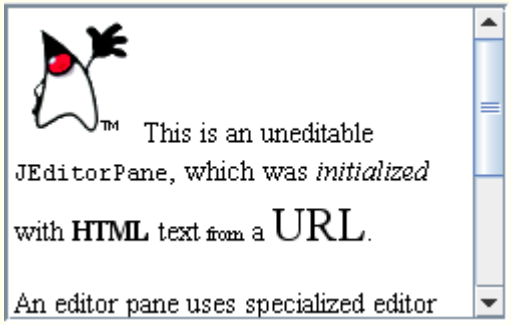
JScrollPane



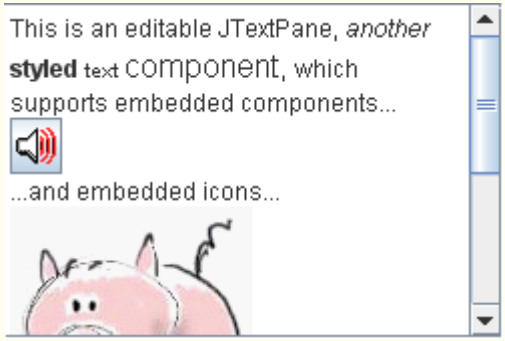
JPanel

JToolBar

JEditorPane



JTextPane



Observer Pattern

Heißt auch Listener oder Publisher-Subscriber Pattern

Anwendungsbeispiel:

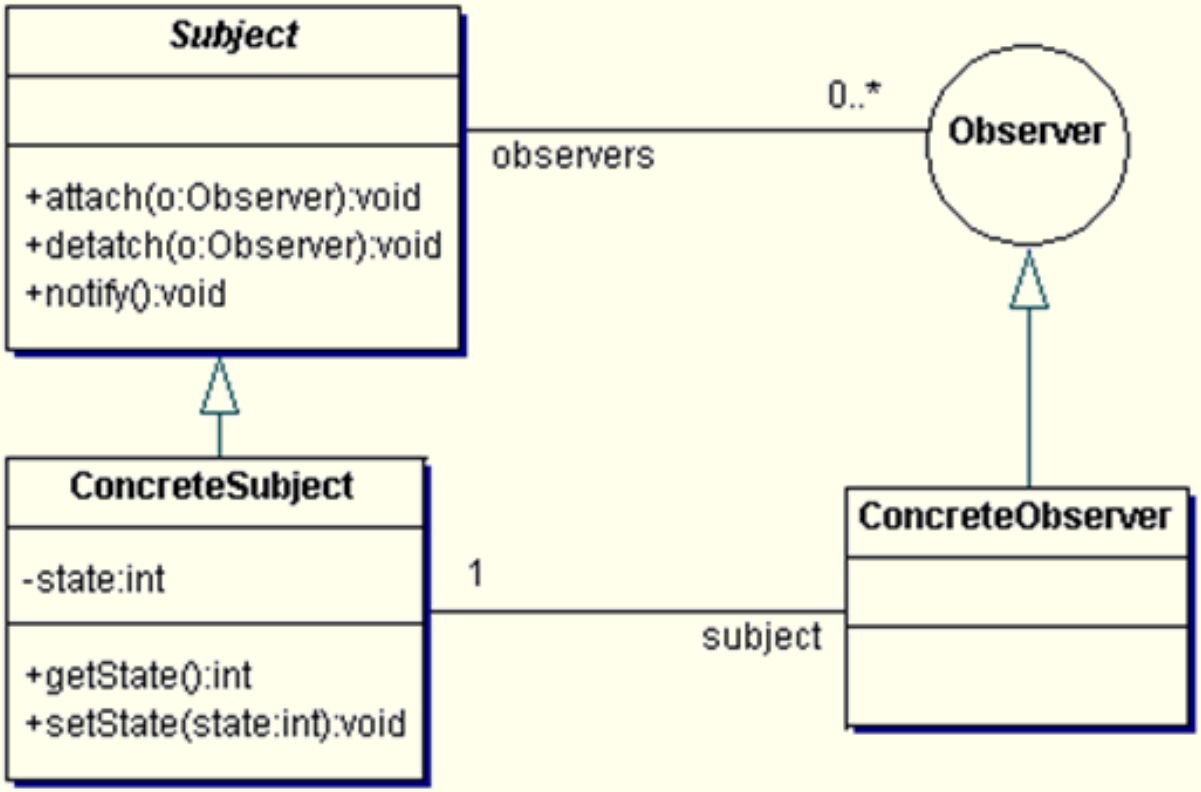
Wenn ein Objekt mitkriegen soll, dass woanders etwas passiert ist, und darauf reagieren soll.

Das Pattern für die ereignisgesteuerte Programmierung:

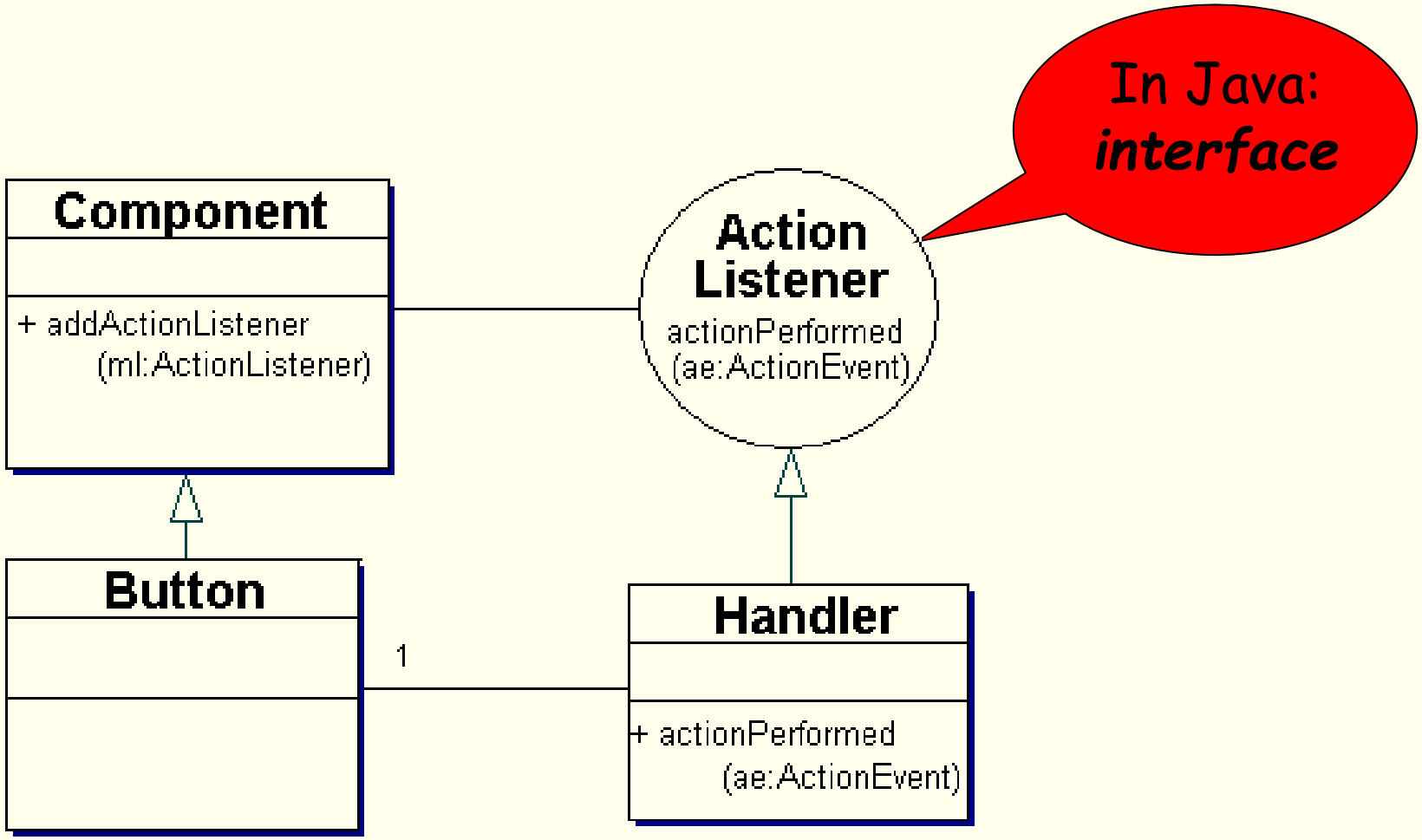
ZB: Ein Button wird geklickt – und ein Controller- bzw. Handler-Objekt muss die entsprechenden Aktionen ausführen.

Observer Pattern

Wenn sich in einem Subjekt der Zustand ändert, werden andere Objekte (die **Observer**) davon informiert.



Beispiel: ActionListener

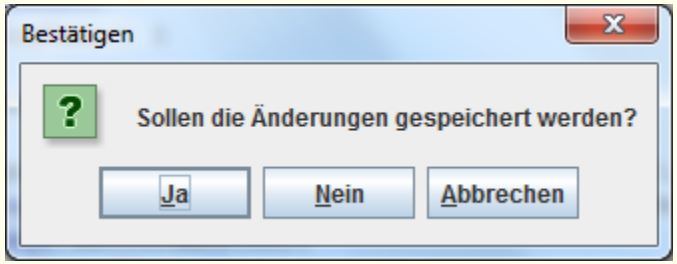


Swing Menüs

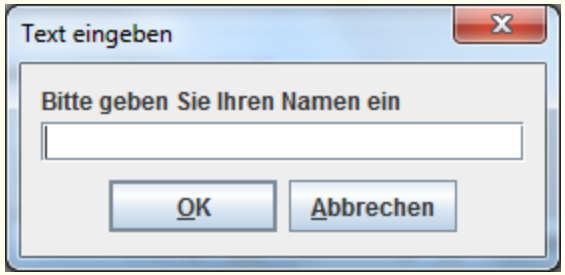
- **JMenuBar**
  - Klasse für die Menüleiste, enthält JMenu-Objekte
- **JPopupMenu**
  - Klasse für Kontextmenüs, enthält JMenu-Objekte
- **JMenu**
  - Menü mit Untermenüs (JMenu), Menüeinträgen (JMenuItem) und Trennlinien (JSeperator)
- **JMenuItem**
  - Menüeintrag
    - ActionListener kann registriert werden
- **JCheckBoxMenuItem, JRadioButtonMenuItem**
  - Menüeintrag als Checkbox bzw. Radiobutton

Swing Dialoge

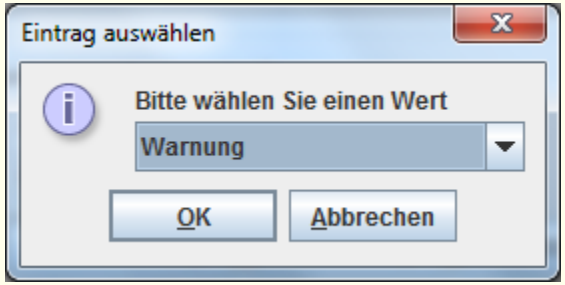
- **JOptionPane**
  - **`showConfirmDialog`** (Messagebox)



  - **`showInputDialog`**
    - mit Eingabefeld

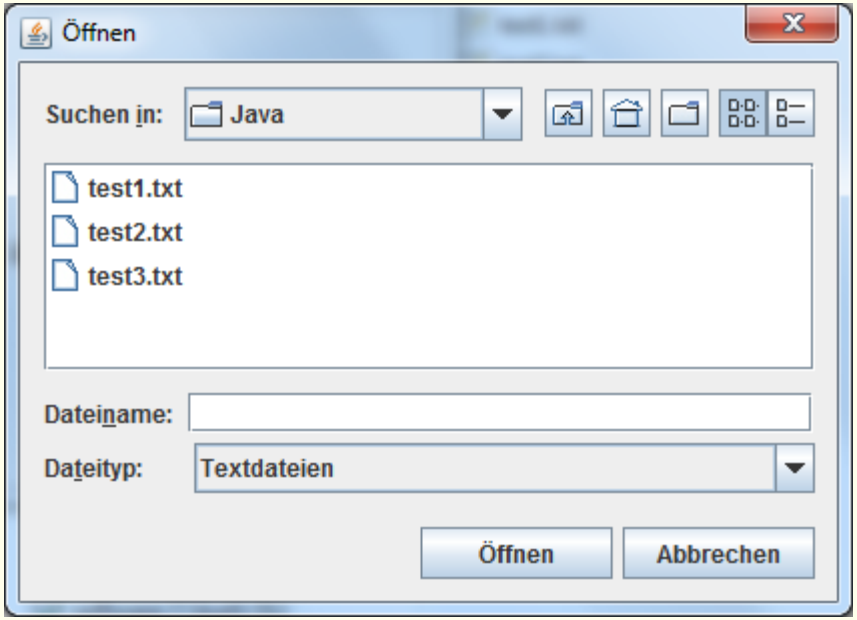


    - mit Combobox



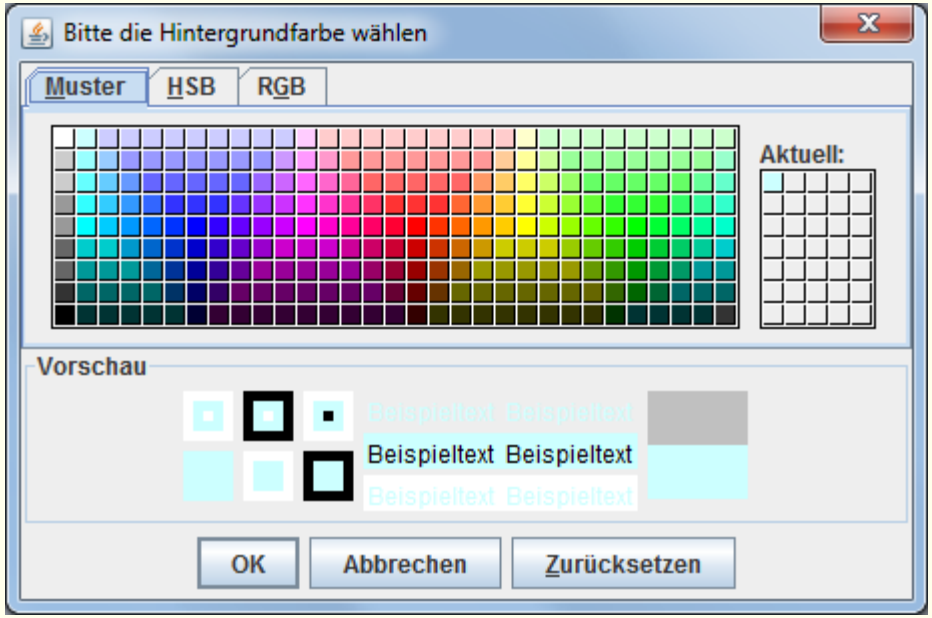
Swing Dialoge

- **JFileChooser**
  - `showOpenDialog`
  - `showSaveDialog`



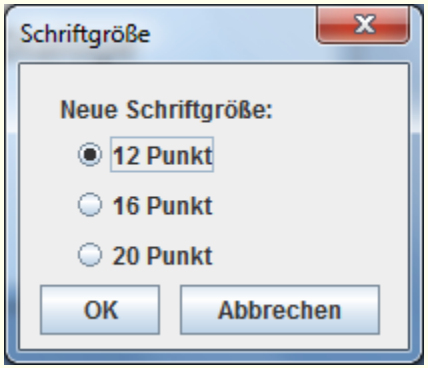
Swing Dialoge

- **JColorChooser**
  - **`showDialog`**
    - Anzeige eines Farbauswahl-Dialogfensters



Swing Dialoge

- **JDialog**
  - Basisklasse für eigene Masken
  - Toplevel Fenster mit Titel und Rahmen
  - hat meistens einen Besitzer (Parent-Frame)
  - 2 Ausführungsarten
    - **modal**: kein anderes Fenster der Applikation kann den Inputfocus erhalten
    - **nicht modal**: andere Fenster der Applikation können den Inputfocus erhalten



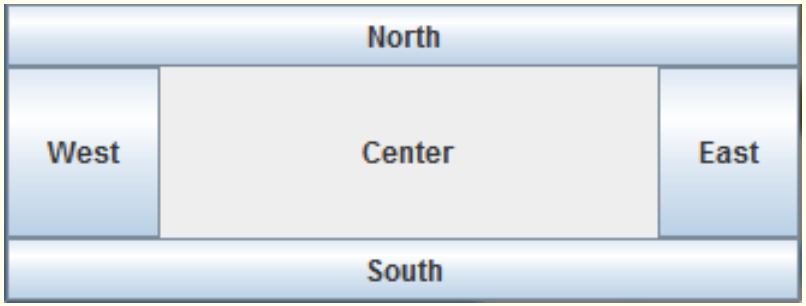
Layout

- **Layout-Manager**
  - ordnen Komponenten eines Containers automatisch an
  - automatisches Anpassen bei Größenänderung
  - **`Container.setLayout(Layout l)`**
    - setzt den Layout-Mananger
    - null um absolute Positionierung zu verwenden
  - **`Container.validate()`**
    - führt die Layoutierung erneut durch
    - Größen etc. werden neu berechnet
  - **`Window.pack()`**
    - setzt die Größe des Fensters so dass alle Komponenten ihre bevorzugte Größe und Layout haben

Layout-Manager

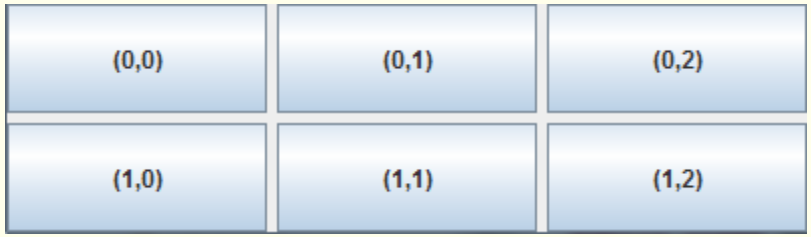
- **FlowLayout**
  - ordnet beliebig viele Komponenten hintereinander an

- **BorderLayout**
  - ordnet 5 Komponenten nach den Himmelsrichtungen an



Layout-Manager

- **GridLayout**
  - ordnet Komponenten in Zeilen und Spalten an



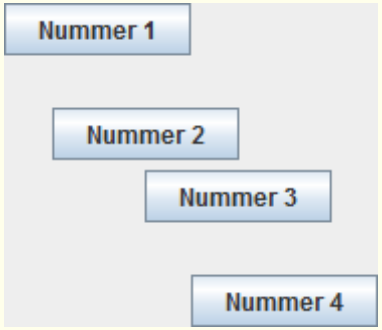
- **GridbagLayout**
  - ordnet Komponenten in überlappenden Zeilen/Spalten an
  - Platzaufteilung nach Gewichtung möglich

Layout-Manager

- **CardLayout**
  - zeigt immer nur 1 von mehreren Komponenten an

- **BoxLayout**
  - ordnet Komponenten horizontal oder vertikal an
  - Komponenten können mit fixem oder flexiblem Abstand positioniert werden

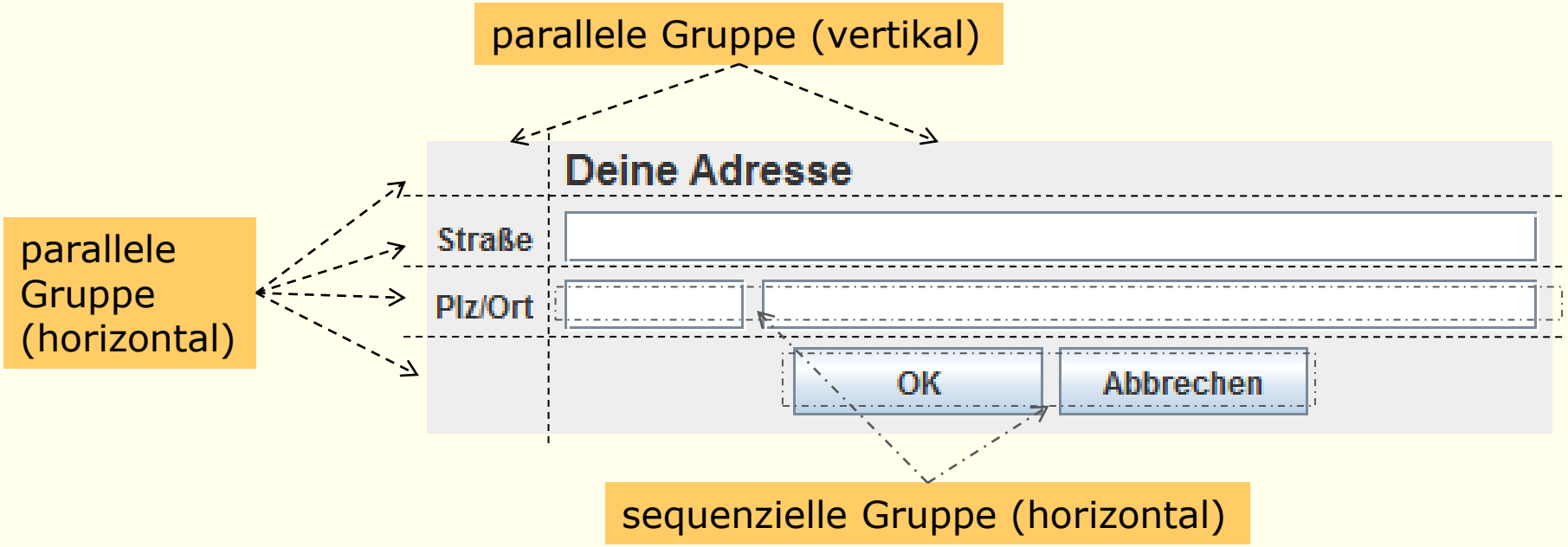
vertikal



horizontal

Layout-Manager

- **GroupLayout**
  - ordnet Komponenten in Gruppen an
    - sequenzielle Gruppen enthalten ihre Komponenten hintereinander (wie im FlowLayout)
    - parallele Gruppen richten ihre Komponenten vertikal oder horizontal aneinander aus



Layout-Manager

- **SpringLayout**
  - verwendet Contstraints für die Anordnung der Komponenten
    - Größe und Position
    - Komponenten können programmatisch ausgerichtet werden
  - ist für den Einsatz im GUI Designer optimiert

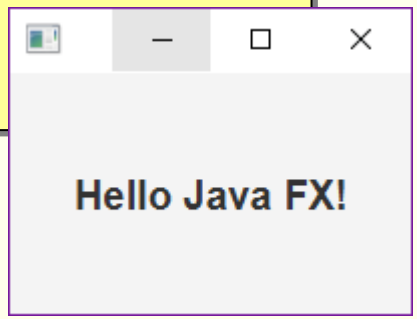
GUI Programmierung mit Java FX

Scene Graph

- **Startpunkt in eine JavaFX Anwendung**
  - hierarchische Knotenstruktur mit allen visuellen Elementen des UI
  - besteht aus
    - Layout Containern
    - Controls
    - Images
    - Texten
    - Shapes
  - Erstellen des Scene Graph
    - programmatisch oder
    - als FXML
  - Styling kann mit CSS erfolgen

JavaFx Beispiel

```
// Content des Scene Graph erzeugen
HBox root = new HBox();
root.setAlignment(Pos.CENTER);
Label lbl = new Label("Hello Java FX!");
lbl.setFont(Font.font("Arial", FontWeight.BOLD, 20));
root.getChildren().add(lbl);
// Scene für den Content erzeugen ...
Scene scene = new Scene(root, 200, 120);
// ... und anzeigen
primaryStage.setScene(scene);
primaryStage.show();
```



- **XML Format zur Definition der JavaFX GUI Komponeten**
  - gibt die Grundstruktur für den Component Tree vor
  - alle JavaFX Controls können analog ihrer API definiert werden
    - die meisten JavaFX Klassen können als Elemente verwendet werden
    - viele Properties der JavaFX Klassen können per Attribut gesetzt werden
  - wird über fx:Controller mit einer Java-Controller-Klasse verknüpft

FXML Syntax

Art	Bedeutung
Instanz-Element	• repräsentiert Instanz einer JavaFX-Klasse • Klasse muss importiert werden <?import ...?> • Elementname mit großem Anfangsbuchstaben
Property-Element	• repräsentiert Property setter, dessen Wert mit Instanz-Element definiert wird • Elementname mit kleinem Anfangsbuchstaben
Property-Attribut	• repräsentiert Property setter, dessen Wert als Zeichenfolge angegeben wird • Attribut in einem Instanzelement
static/attached Property	• definiert Property für ein Objekt in seinem Parent-Container

FXML Syntax

- **Resolution Operators**
  - Operator, der einem Attribut-Wert eine besondere Bedeutung gibt

Operator	Bedeutung	Beispiel
@	relativer Pfad	"@logo.png"
%	Key einer Zeichenfolge aus einem Resource Bundle	"%login.title"
$	Verweis auf Variable	"$grpGender"
${}	Binding-Ausdruck	"${txtName.text}" "${'Mehrzeiliger\nText'}

Wenn der Wert mit einem der Operatoren beginnt, muss er mit \ entwertet werden:

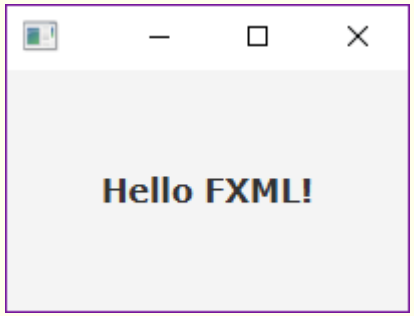
```
\% \$ \@
```

FXML Beispiel

```
<HBox fx:controller="hello2.HelloFxController"
  alignment="CENTER" >
  <children>
    <Label text="Hello FXML!" fx:id="lblTitle" />
  </children>
</HBox>
```

```
public class HelloFxController {
  @FXML  private Label lblTitle;
  @FXML  private void initialize() {
  lblTitle.setFont(Font.font("Verdana",
    FontWeight.BOLD, 16));
  }
}
```

```
HBox root = (HBox) FXMLLoader.load(
   getClass().getResource("HelloJavaFx.fxml"));
Scene scene = new Scene(root, 200, 120);
primaryStage.setScene(scene);
primaryStage.show();
```



JavaFx Layout Container

Container	Bedeutung
GridPane	Child-Knoten werden in Zeilen und Spalten angeordnet, Zellen können sich über mehrere Zeilen und Spalten erstrecken
BorderPane	Kann bis zu 5 Child-Knoten in 5 Bereichen (top, right, bottom, left, center) enthalten
AnchorPane	Beliebige Child-Knoten, die an den Rändern angedockt werden können
FlowPane	Beliebige Child-Knoten, die nebeneinander angezeigt werden, Umbruch falls zu breit
TilePane	Wie FlowPane mit gleich großen Knoten
StackPane	Die Child-Knoten werden übereinander gelegt
TabPane	Beliebig viele Tab-Items, von denen jeweils eines aktiv ist

JavaFx Layout Container

Container	Bedeutung
ScrollPane	Ein Child-Knoten und Scrollbars
TitledPane	Titel und 1 Child-Knoten (einklappbar)
Accordion	Eine Gruppe von TitledPanes, von denen immer nur eines geöffnet ist
SplitPane	Teiler (vertikal oder horizontal) mit 2 Child-Knoten
ToolBar	Leiste für Tool-Bar Items
HBox	Die Child-Knoten werden in einer horizontalen Linie angeordnet
VBox	Die Child-Knoten werden in einer vertikalen Linie angeordnet

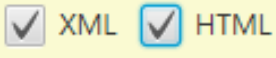
JavaFx Controls

- **Interaktion mit dem Benutzer**
  - Einfache Controls, z.B.
    - Label, TextBox, Button, usw.
  - Auswahlfelder für einen Wert
    - ComboBox, ChoiceBox
  - Listenorientierte Controls, z.B.
    - ListView, TableView
  - Hierarchische Controls
    - TreeView
  - Menüs, Contextmenüs, Toolbars

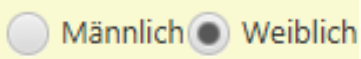
JavaFx Controls

Button

CheckBox



RadioButton

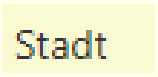


TextField

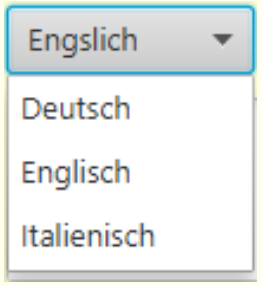
PasswordField

TextArea

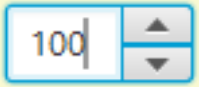
Label



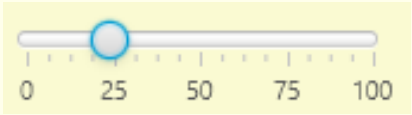
ComboBox



Spinner



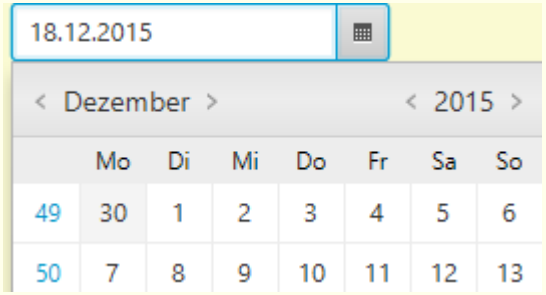
Slider



ProgressBar

ProgressIndicator

DatePicker



JavaFX Observable Property

- **hat 3 Bestandteile**
  - T getXxx(): getter für den Wert der Property
  - void setXxx(T value): setter für den Wert der Property
  - ObservableValue<T> xxxProperty(): die Property als solche
    - kann für Änderungsbenachrichtigung verwendet werden

```
txtName.textProperty()
        .addListener(new ChangeListener<String>() {
   public void changed(ObservableValue<? extends String> o,
                String oldValue, String newValue) {
        if(newValue != null && !newValue.isEmpty()){
                System.out.println("New Value OK!");
        }
   }
});
```

JavaFx Menüs

- **MenuBar**
  - Klasse für die Menüleiste, enthält Menu-Objekte
- **PopupMenu**
  - Klasse für Kontextmenüs, enthält Menu-Objekte
- **Menu**
  - Menü mit Untermenüs (Menu), Menüeinträgen (MenuItem)
- **MenuItem**
  - Menüeintrag mit ActionHandler
  - Spezialisierungen:
    - CheckMenuItem: Checkbox
    - RadioMenuItem: Radiobutton
    - SeperatorMenuItem: Trennlinie

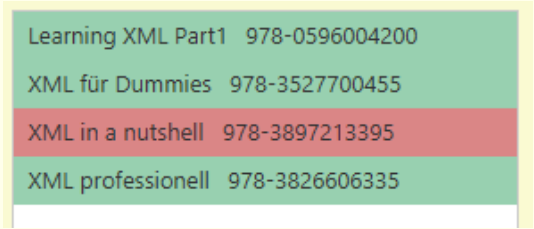
Komplexe JavaFx Controls

TableView

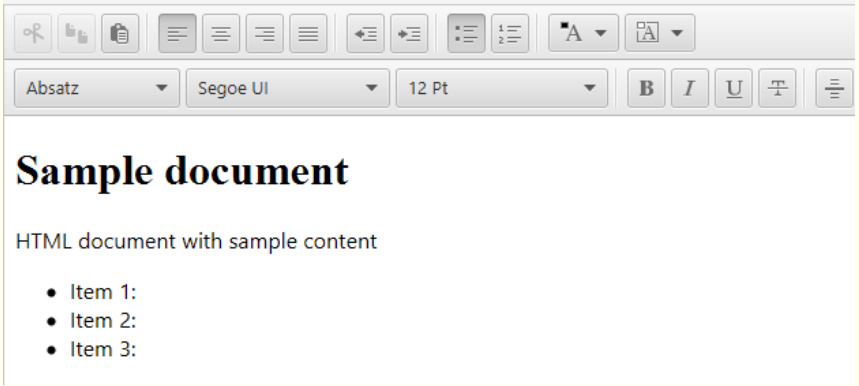
Title	ISBN	Price	Stock	In Stock
Learning XML Part1	978-0596004200	10.5	10	true
XML für Dummies	978-3527700455	8.5	10	true
XML in a nutshell	978-3897213395	21.5	0	false
XML professionell	978-3826606335	9.5	10	true

WebView

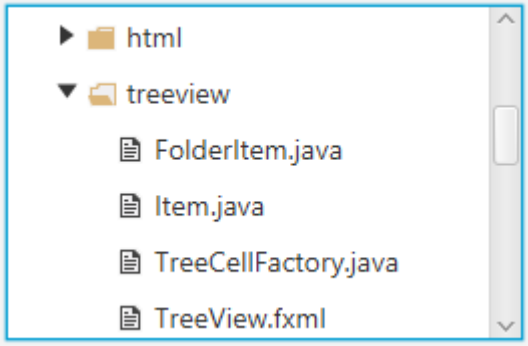
ListView



HTMLEditor

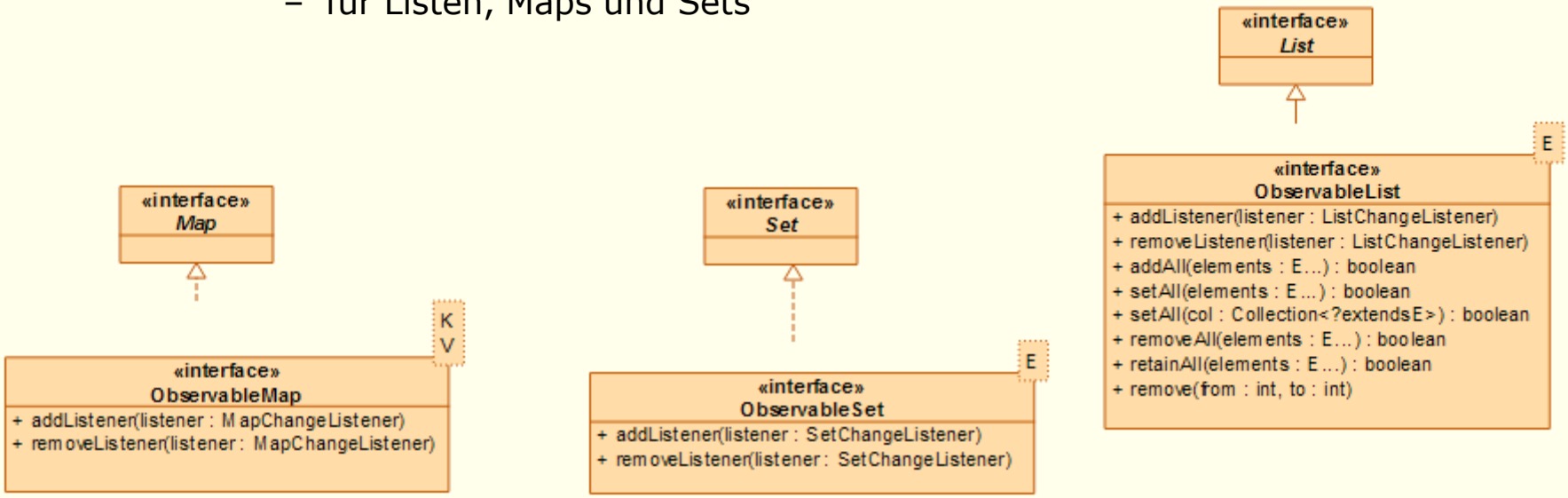


TreeView

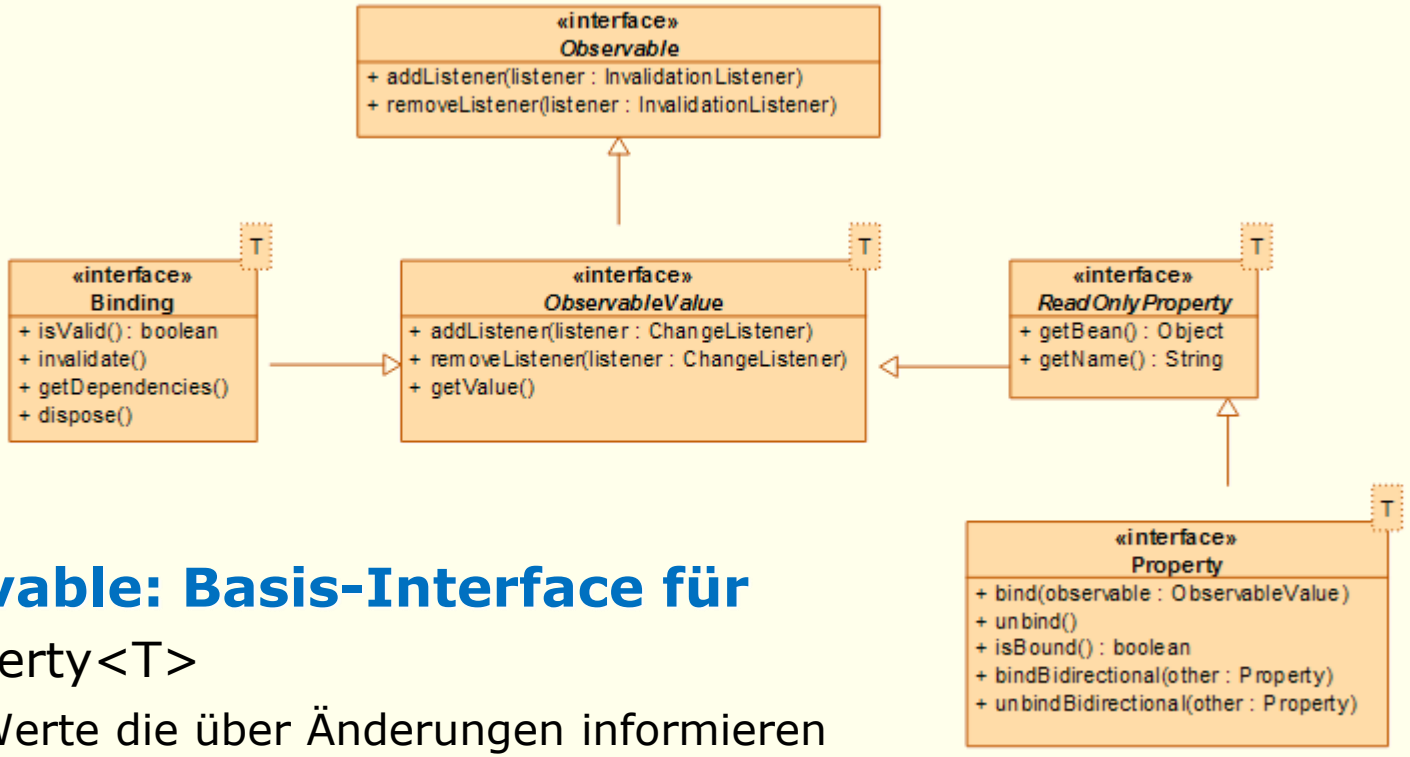


Property Binding

- **Automatischer Austausch von Werten zwischen zwei Java Objekten**
  - Basiert auf JavaFX Observable Properties und deren Benachrichtigungssystem
  - Handling über gemeinsame Interfaces
    - Observable für Properties und Bindings
    - ObservableList, ObservableMap, ObservableSet
      - für Listen, Maps und Sets



Property Binding – Observable

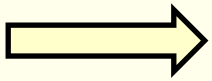


- **Observable: Basis-Interface für**
  - Property<T>
    - Werte die über Änderungen informieren
    - für Properties der eigenen Model-Klassen
  - Binding<T>:
    - Werte die an andere Werte gebunden sind
    - um eine Property (*von Objekt A*) an eine oder mehrere andere Properties (*von Objekt B*) zu binden

Property Binding – Beispiel

```
// Define properties
IntegerProperty num1 = new SimpleIntegerProperty(10);
IntegerProperty num2 = new SimpleIntegerProperty(2);
// Bind the result to the properties
NumberBinding result = Bindings.multiply(num1, num2);
// Get notified if value changes
result.addListener((obs, oldValue, newValue) -> {
    System.out.printf("Value changed to %s\n", newValue);
});

System.out.printf("Result: %s\n", result.getValue() );
num1.set(5); // Change the value of a property
System.out.printf("Result: %s\n", result.getValue() );
```



```
Result: 20
Value changed to 10
Result: 10
```

Property Binding – Objekte instanziieren

- **Property<T>**
  - fertige Implementierungen für Grunddatentypen sowie String und Object (SimpleXxxProperty)
  - eigene Implementierungen durch Ableiten von XxxPorpertyBase
- **Binding<T>**
  - Factory-Methoden der Bindings-Klasse
    - Arithmethische Operationen (add, multiply)
    - Logische Operationen (and, or, not)
    - Vergleiche (isNull, equal, greaterThan, ...)
- **ObservableList<T>**
  - Factory-Methoden der FXCollections-Klasse
    - observableArrayList, observableHashMap, ...