

# Objektorientierte Modellierung und Programmierung

Dr. Christian Schönberg

# GUI-Frameworks und - Anwendungen I

- JavaFX
  - Controls
  - Layout
  - Fenster

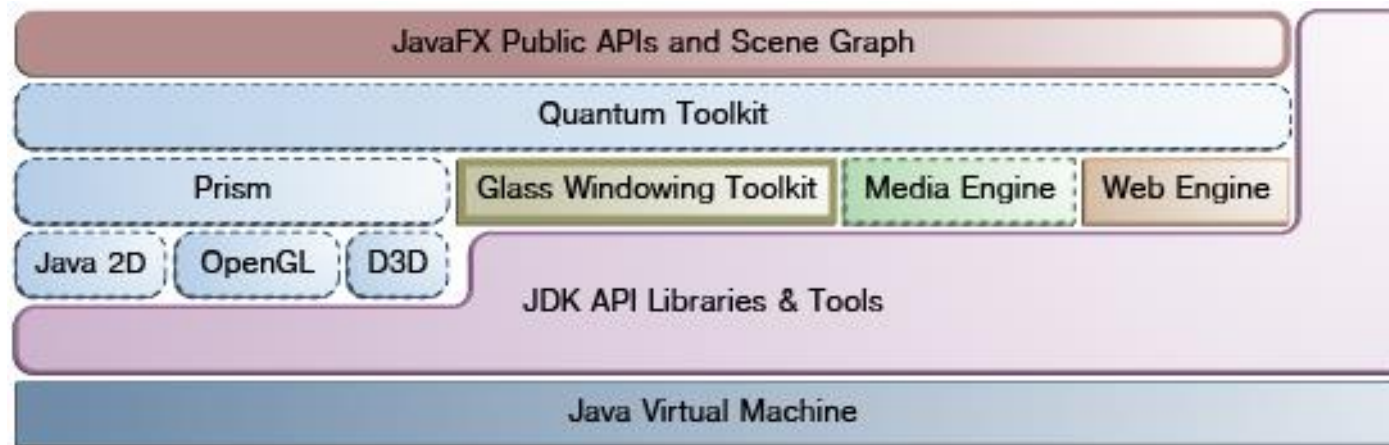
- AWT (Abstract Window Toolkit):
  - Zeichnen von grafischen Primitiven (Linien, Polygone, Text, ...)
  - Delegation-based Ereignisbehandlung (Maus, Tastatur, ...)
  - GUI-Komponenten (Fenster, Buttons, ...)
  - Layout-Manager
  - Weiteres (Bitmaps, Sound, ...)
  - Kennzeichen: Heavyweight-GUI-Komponenten
- Swing: seit Java 1.2
  - baut auf AWT auf
  - Erweiterung der GUI-Komponenten-Sammlung
  - Model-View-Controller-Prinzip
  - Pluggable Look-and-Feel-Prinzip
  - Kennzeichen: Lightweight-GUI-Komponenten
- JavaFX: seit Java 8, aber ab Java 11 nur noch als Zusatzmodul
- Installation: <https://openjfx.io/openjfx-docs/>

# JavaFX: Eigenschaften

- FXML: UI-Gestaltung per XML und CSS3
- Scene Builder: interaktives Tool für UI-Gestaltung
- WebView: integrierter Web-Browser (→ WebKit)
- Swing-Interoperabilität (→ **SwingNode**-Klasse)
- Build-in-UI-Controls: alle gängigen UI-Komponenten integriert
- CSS3: Komponentendesign via CSS3
- 2D-/3D-Graphics: integrierte 2D- und 3D-Grafik-Objekte
- Canvas API: Zeichnen von Grafik-Primitiven
- Printing-API: API zum Drucken
- Multimedia: Images, Audio, Video, Animationen
- ...

# JavaFX: Architektur

- Prism: High-performance graphics engine
- Glass: effizientes Window-System
- Quantum Toolkit: Verbindung der darunter liegenden Schichten



<http://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-architecture.htm>

# JavaFX: Beispiel

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

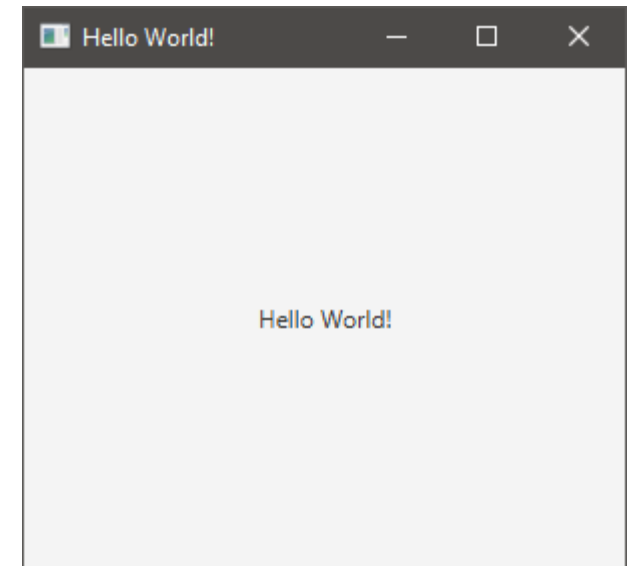
public class HelloWorld extends Application {

    public static void main(String[] args) {
        Launch(args);
    }

    ...
}
```

# JavaFX: Beispiel

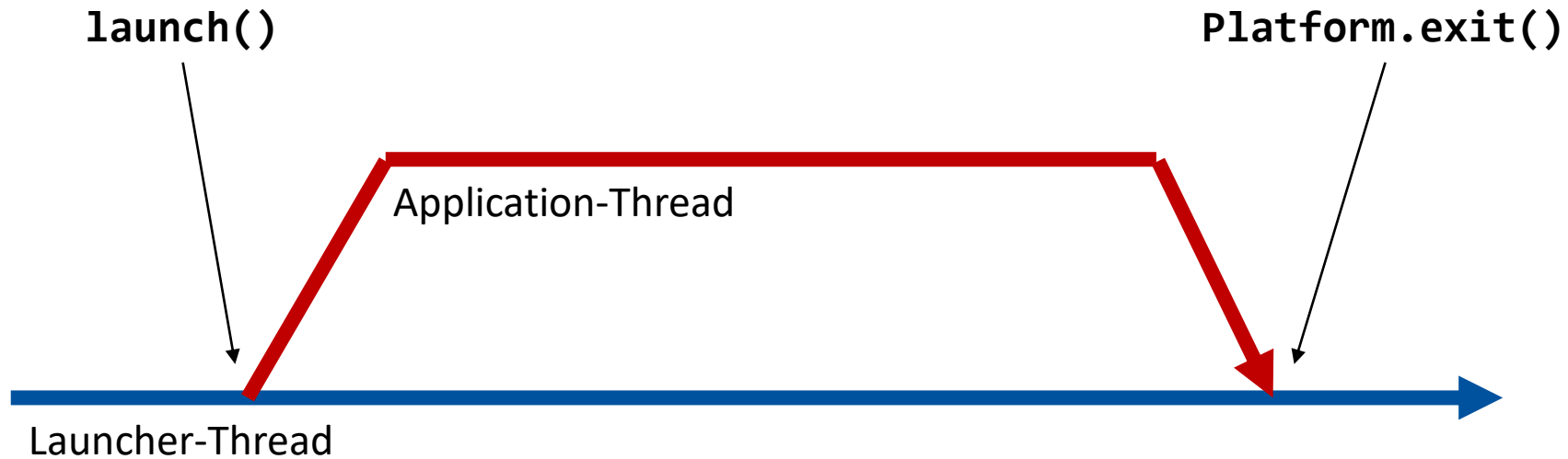
```
...  
  
@Override  
public void start(Stage primaryStage) {  
    Label label = new Label("Hello World!");  
  
    StackPane root = new StackPane();  
    root.getChildren().add(label);  
  
    Scene scene = new Scene(root, 300, 250);  
  
    primaryStage.setTitle("Hello World!");  
    primaryStage.setScene(scene);  
    primaryStage.show();  
}  
  
}
```





- Ableiten von **Application**
- Aufruf von **launch** (→ **main**-Thread)
- Überschreiben der Methoden
  - **public void init()** (→ optional)
    - Wird vom JavaFX-Launcher-Thread ausgeführt
    - Keine UI-Operationen!
    - Aufrufparameter auslesen
  - **public void start(Stage stage)**
    - Wird vom JavaFX-Application-Thread ausgeführt
    - **stage**: Intern erzeugtes Haupt-Fenster
  - **public void stop()** (→ optional)
    - Wird vom JavaFX-Application-Thread ausgeführt
- Beenden mit:
  - **Platform.exit();**

# JavaFX Threads



Der Application-Thread ist für alles zuständig, was direkt mit der GUI zusammenhängt:

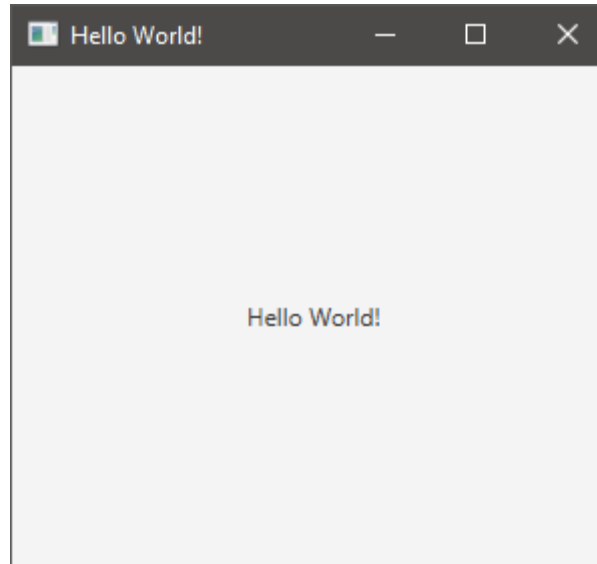
- grafische Elemente erzeugen bzw. anzeigen
- Animationen
- Events (z.B. Reagieren auf Eingaben)

→ es dürfen keine langwierigen Berechnungen o.ä. auf dem Application-Thread ausgeführt werden, da sonst die GUI „hängenbleibt“

→ Details später beim Thema Multithreading

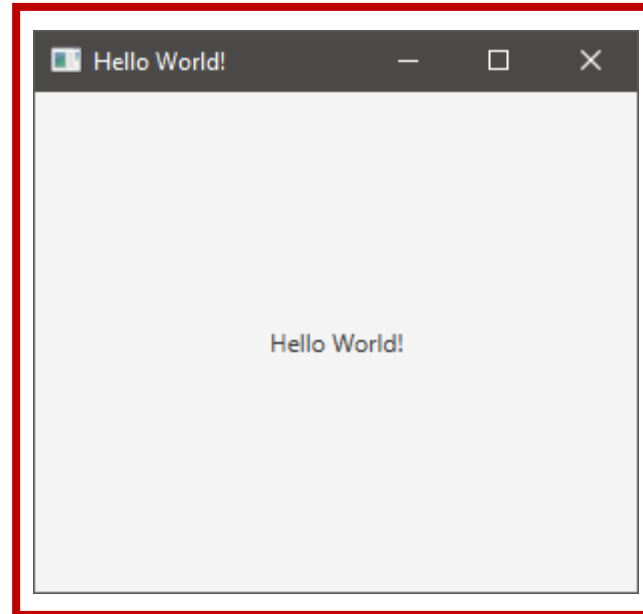
- **Stage** (Bühne)
- **Scene** (Leinwand)
  - Verwaltet **SceneGraph**
    - Gewurzelter Baum
    - Aufgebaut aus **Node**-Objekten
    - **Parent**-Nodes: **Node**-Objekte, die weitere **Node**-Objekte enthalten können
  - Bindeglied zwischen **Stage** und **SceneGraph**
  - Fenstergrößenveränderung → Information an Wurzelknoten
- **Node**-Typen
  - GUI-Komponenten
  - Container
  - Grafik-Objekte

# JavaFX Anwendung: Aufbau



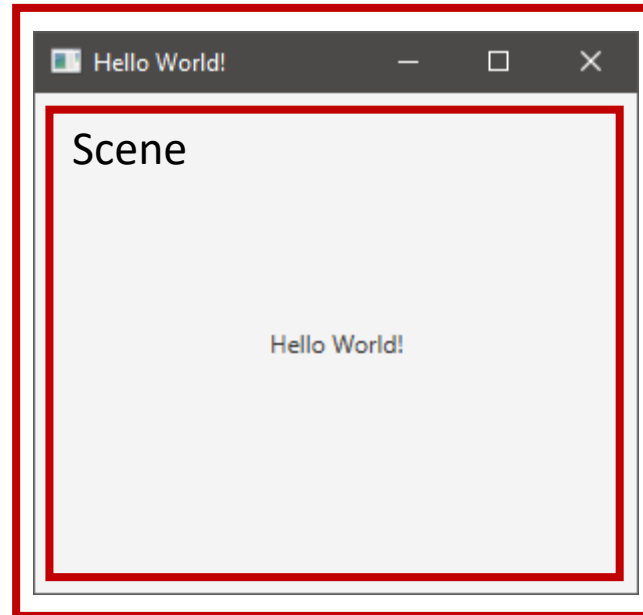
# JavaFX Anwendung: Aufbau

Stage



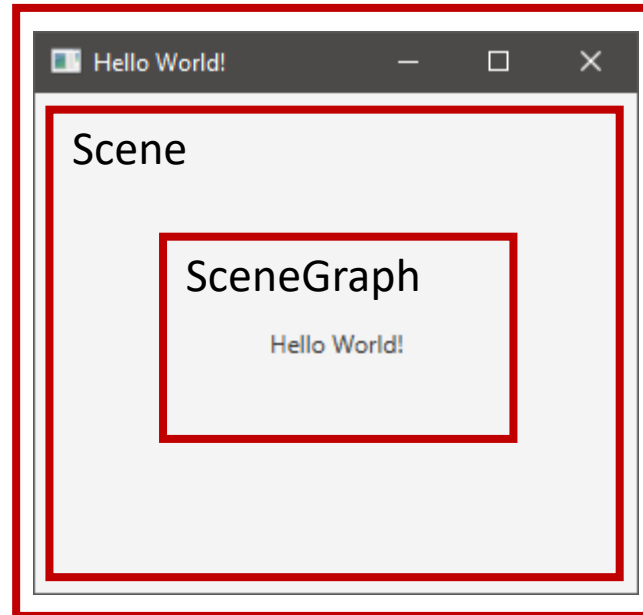
# JavaFX Anwendung: Aufbau

Stage

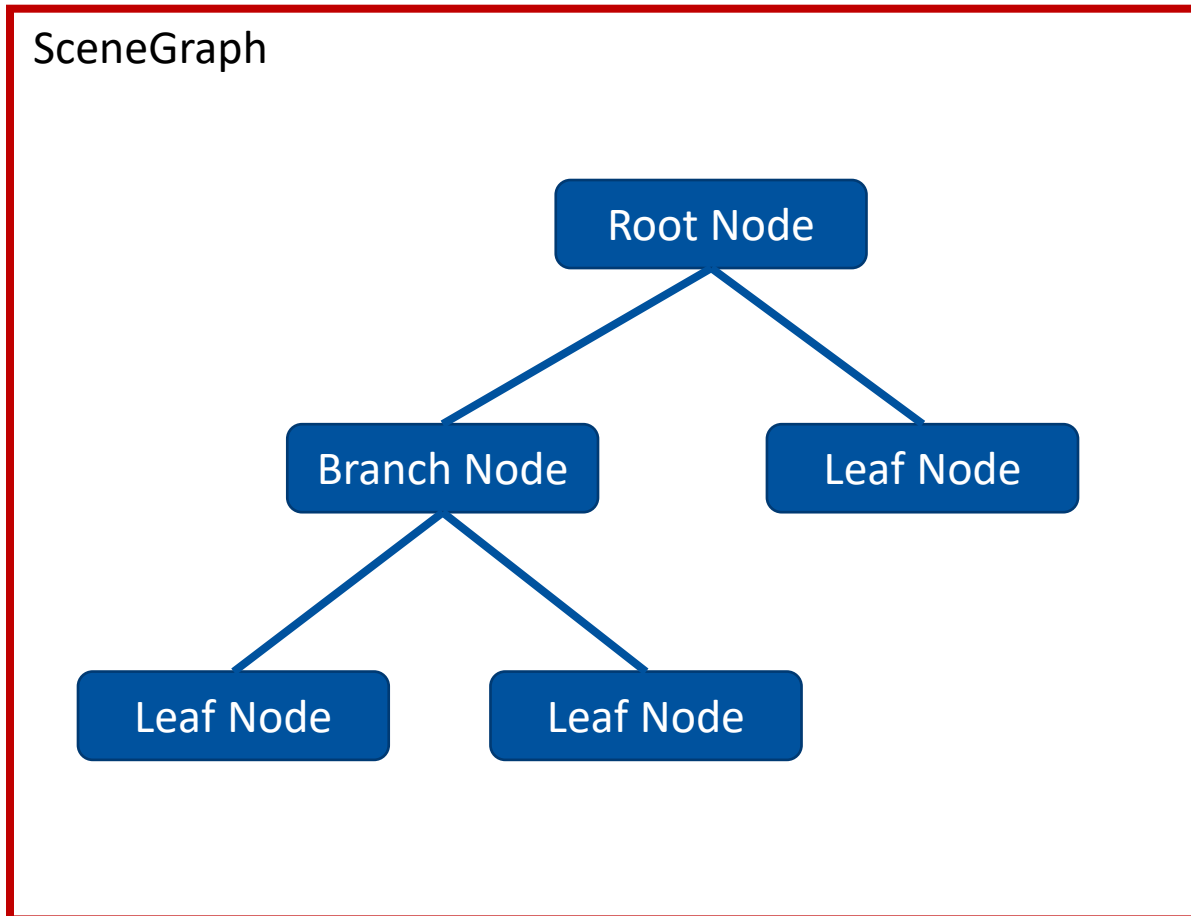


# JavaFX Anwendung: Aufbau

Stage



# JavaFX Anwendung: Aufbau





# JavaFX: Wichtige Klassen

- **Stage:** Fenster
- **Scene:** Leinwand
- **Node:** Elemente des Szenegraphen
  - **Parent:** Basisklasse für Container-Klassen
    - **Group:** Zusammenfassung von Nodes für Transformationen oder Effekte auf Gruppen von Objekten
    - **Region:** Basisklasse für UI-Komponenten und Layouts
      - **Control:** Basisklasse für UI-Komponenten
      - **Pane:** Basisklasse für Layouts
      - **Chart:** Basisklasse für Charts
    - **WebView:** verwaltet WebEngine

# Wiederholung: Anonyme Klassen

- Anonyme Klassen instanziiieren Oberklassen oder Interfaces, ohne einen eigenen (Unter-) Klassennamen anzugeben
- Sie sind geeignet für einfache Klassen, die nur einmal gebraucht werden
- Anonyme Klassen können beliebige Attribute und Methoden definieren, aber sie werden immer dynamisch an eine Objektvariable vom Typ der Oberklasse gebunden
  - Zugriff nur auf die Attribute und Methoden der Oberklasse
  - zusätzliche Attribute und Methoden nützlich als Hilfsattribute und -methoden

# Anonyme Klassen: Beispiel

```
List<Fruit> basket = new ArrayList<>();  
basket.add(new Apple());  
basket.add(new Banana());  
basket.add(new Pineapple());  
basket.add(new Fruit() {  
    @Override  
    public String getOrigin() {  
        return "Malta";  
    }  
});
```

Klassendefinition direkt nach dem Konstruktoraufruf in { }.

# Wiederholung: Lambda-Ausdrücke



- **Lambda-Ausdrücke** sind keine anonymen Klassen, sondern **anonyme Methoden**
- Sie werden aber eingesetzt, um die Implementierung von funktionalen Interfaces durch einen einfachen, kompakten Ausdruck zu ersetzen
  - funktionale Interfaces sind Interfaces, die nur eine einzige nicht-statische und nicht-default Methode definieren
  - Instanzen von funktionalen Interfaces können direkt durch einen Lambda-Ausdruck ersetzt werden
  - dabei wird die Lambda-Funktion auf die eine Methode des Interfaces abgebildet
- Aufbau: `FunctionalInterface instance = (parameters) -> expression/block;`

# Beispiel: Lambda-Ausdruck

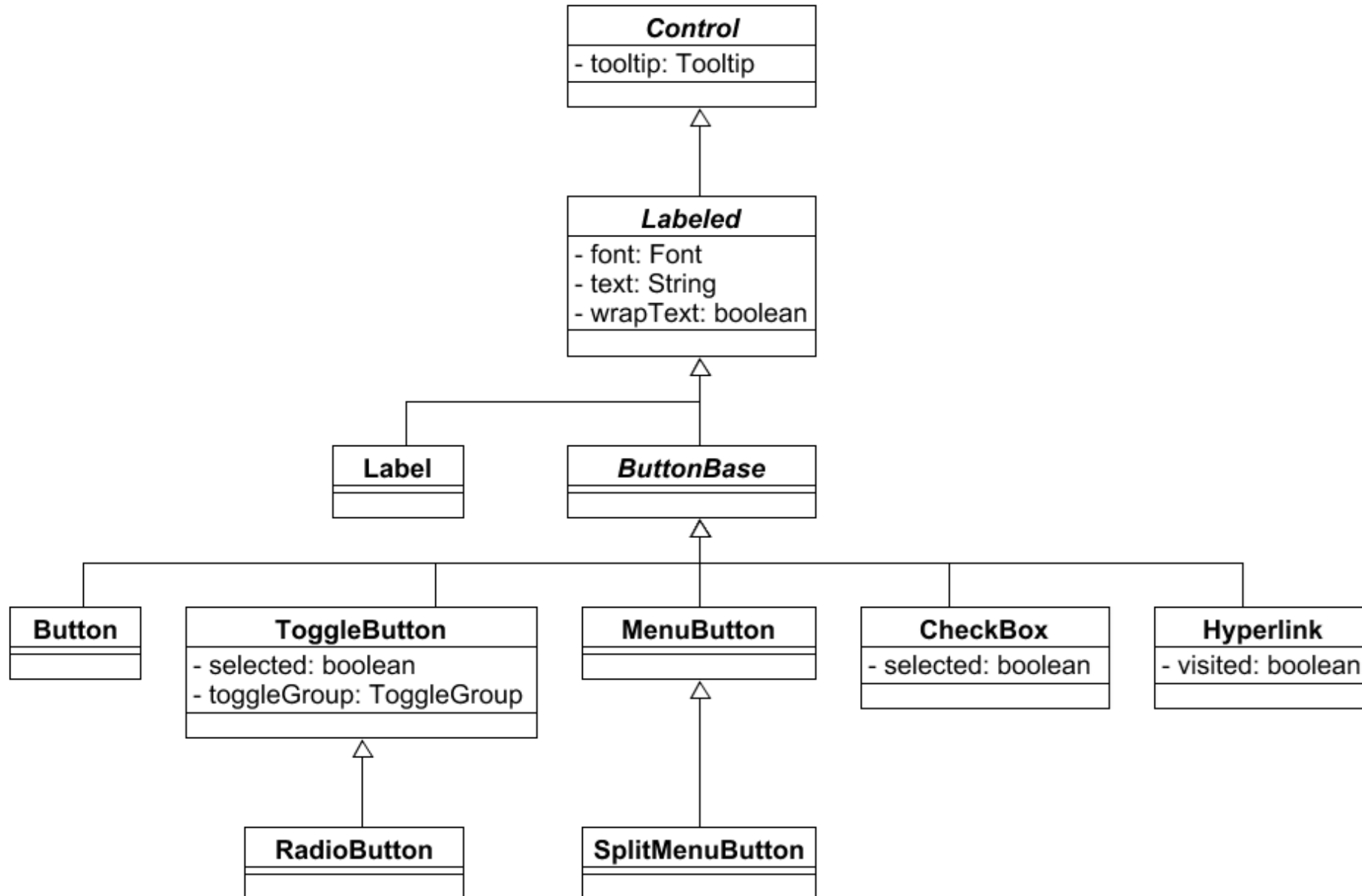
Collections
<u>+ sort(list: List&lt;T&gt;, comp: Comparator&lt;T&gt;)</u>

«interface» Comparator<T>
+ compare(o1: T, o2: T): int

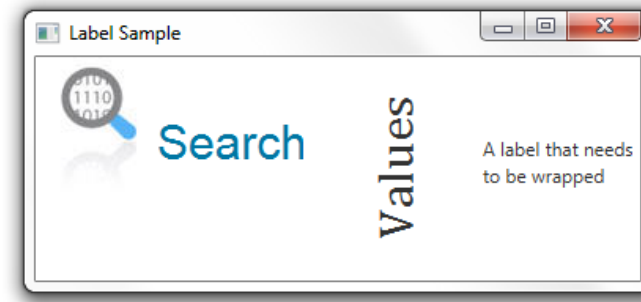
```
List<Integer> list = new ArrayList<>();  
list.add(3);  
list.add(1);  
list.add(2);  
list.add(4);  
  
Collections.sort(list, (i1, i2) -> Integer.compare(i1, i2));  
System.out.println(list); // [1, 2, 3, 4]  
  
Collections.sort(list, (i1, i2) -> Integer.compare(i1, i2) * -1);  
System.out.println(list); // [4, 3, 2, 1]
```

# Controls

# Label und Buttons



- Beinhaltet Text + Node-Objekt

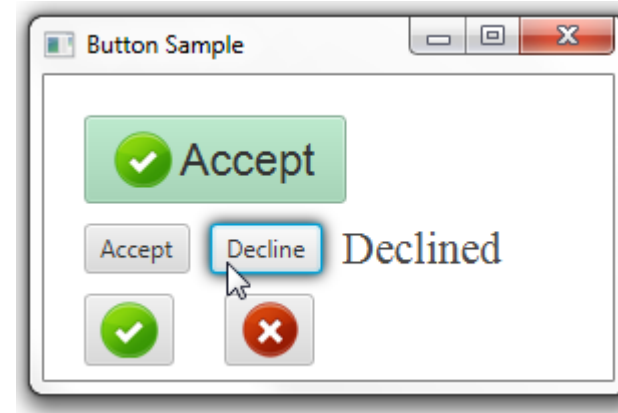


<https://docs.oracle.com/javase/8/javase-books.htm>

```
Label label = new Label("Hello World!");  
Image image = new Image("world.png");  
label.setGraphic(new ImageView(image));  
label.setFont(new Font("Arial", 20));  
label.setContentDisplay(ContentDisplay.TOP);  
label.setGraphicTextGap(30);  
label.setTextAlignment(TextAlignment.CENTER);  
label.setRotate(45);  
label.setText("Ersetzen wir den Text gegen einen längeren Text, "  
+ " ist es sinnvoll, diesen Text bei Bedarf zu umbrechen.");  
label.setTextWrap(true);
```



# Button

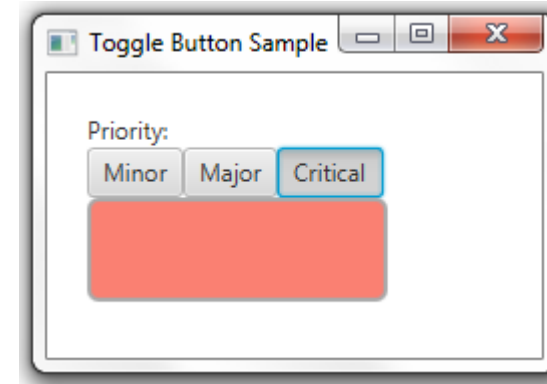


<https://docs.oracle.com/javase/8/javase-books.htm>

```
Button button = new Button("Bitte klicken Sie mich!");
button.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        button.setText("Danke!");
    }
});
Button defaultButton = new Button("Default");
defaultButton.setDefaultButton(true); // -> VK_ENTER-Taste
Button button3 = new Button("Hello World!");
Image image = new Image("world.png");
// Image image = new Image(
//     getClass().getResource("/world.png").toString());
button3.setGraphic(new ImageView(image));
```

# ToggleButton

- Button mit selected-Zustand
- Gruppierung möglich

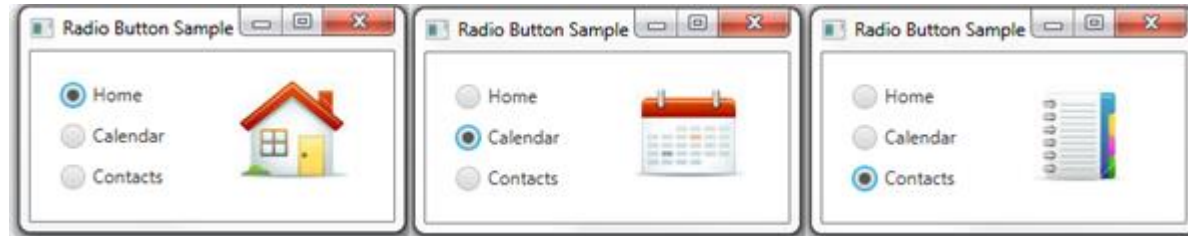


<https://docs.oracle.com/javase/8/javase-books.htm>

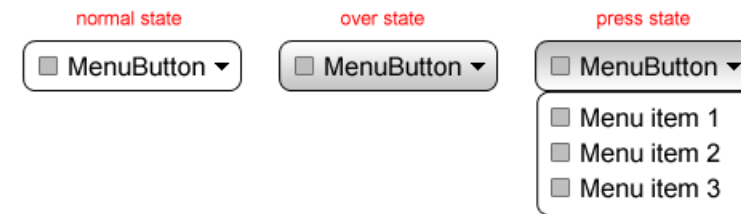
```
final ToggleGroup group = new ToggleGroup();
ToggleButton tb1 = new ToggleButton("Minor");
tb1.setToggleGroup(group);
tb1.setSelected(true);
ToggleButton tb2 = new ToggleButton("Major");
tb2.setToggleGroup(group);
tb1.setUserData(Color.LIGHTGREEN);
tb2.setUserData(Color.LIGHTBLUE);
group.selectedToggleProperty().addListener((observable, oldValue, newValue) -> {
    if (newValue == null) {
        rect.setFill(Color.WHITE);
    } else {
        rect.setFill((Color) group.getSelectedToggle().getUserData());
    }
});
```

# RadioButton

- Abgeleitet von ToggleButton
- Anderes Aussehen
- Immer ein Button einer Gruppe selektiert



<https://docs.oracle.com/javase/8/javase-books.htm>

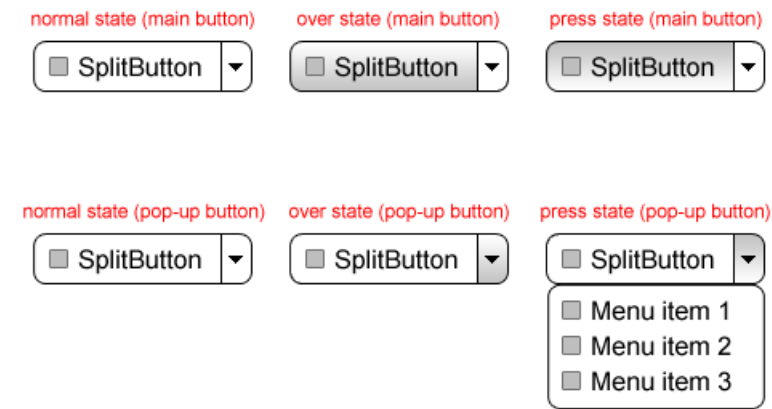


## ■ Button mit ContextMenu

```
MenuButton m = new MenuButton("Kartoffelsalate");
m.setPopupSide(Side.RIGHT);
MenuItem menuItem1 = new MenuItem("Bayerische Art");
menuItem1.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        System.out.println("Ich mag ihn auf bayerische Art");
    }
});
MenuItem menuItem2 = new MenuItem("Rheinische Art");
menuItem2.setOnAction((event) ->
    System.out.println("Ich mag ihn auf rheinische Art"));
m.getItems().addAll(menuItem1, menuItem2);
```

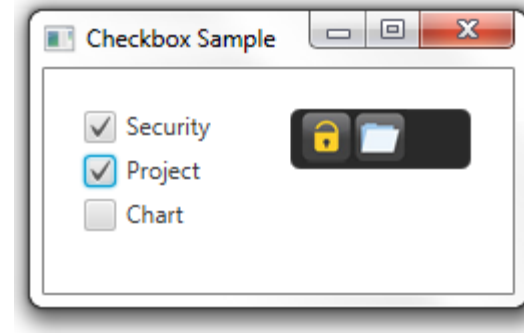
# SplitMenuButton

- Analog zu MenuButton
- Aber mit Action-Bereich



```
SplitMenuButton m = new SplitMenuButton();
m.setText("Kartoffelsalate");
m.setPopupSide(Side.RIGHT);
m.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        System.out.println("Ich mag Kartoffelsalat");
    }
});
MenuItem menuItem1 = new MenuItem("Bayerische Art");
menuItem1.setOnAction(// ... );
MenuItem menuItem2 = new MenuItem("Rheinische Art");
menuItem2.setOnAction(// ... );
m.getItems().addAll(menuItem1, menuItem2);
```

- Kontrollkästchen
- Drei Zustände
  - checked, unchecked, undefined



<https://docs.oracle.com/javase/8/javase-books.htm>

```
CheckBox checkBox = new CheckBox("I'm undecided");
checkBox.setAllowIndeterminate(true);
checkBox.setOnAction((event) -> {
    if (checkBox.isIndeterminate()) {
        checkBox.setText("I'm undecided");
    } else if (checkBox.isSelected()) {
        checkBox.setText("I agree");
    } else if (!checkBox.isSelected()) {
        checkBox.setText("I disagree");
    }
});
```

# Hyperlink

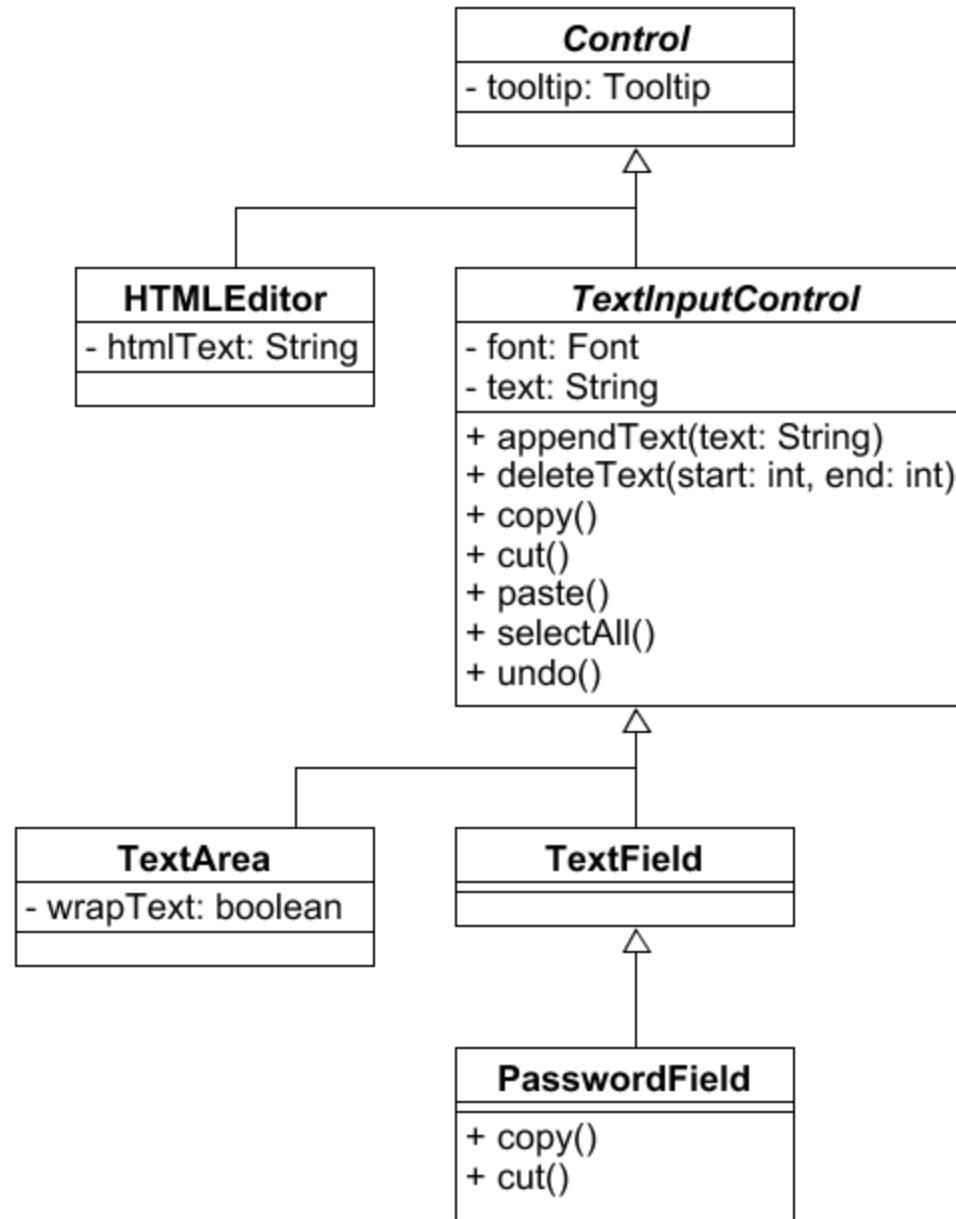
<http://example.com> — unvisited link

<http://example.com> — link is clicked

<http://example.com> — visited link

<https://docs.oracle.com/javase/8/javase-books.htm>

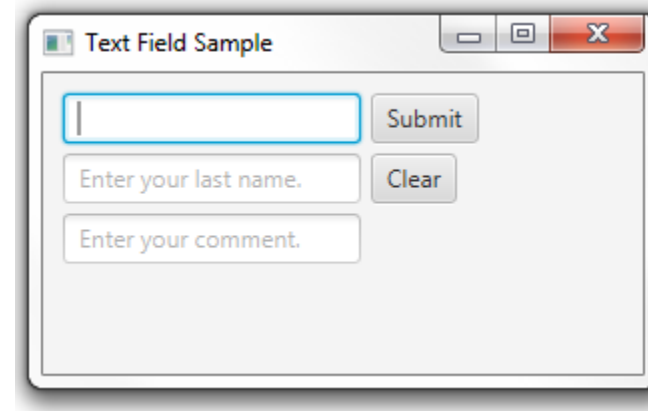
```
Hyperlink hyperlink = new Hyperlink("www.klickmich.de");  
hyperlink.setOnAction((event) -> hyperlink.setText("www.schongeklickt.de"));
```





# TextField

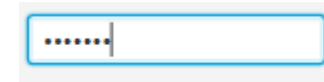
- Einzeilige Texte
- Keine direkte Beschränkung



<https://docs.oracle.com/javase/8/javase-books.htm>

```
TextField textField = new TextField();  
textField.setPromptText("Bitte hier etwas eingeben...");  
textField.setOnAction((event) ->  
    System.out.println("Eingabe: " + textField.getText()));
```

# PasswordField

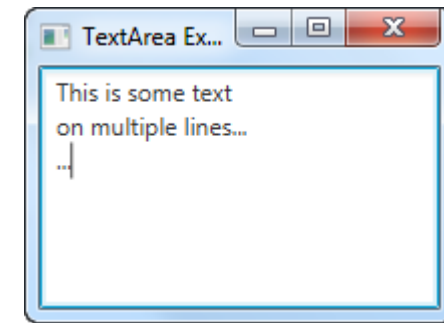


<https://docs.oracle.com/javase/8/javase-books.htm>

- Eingabe wird maskiert

```
PasswordField passwordField = new PasswordField();  
passwordField.setPromptText("Bitte Passwort eingeben!");  
passwordField.setOnAction((event) ->  
    System.out.println(passwordField.getText()));
```

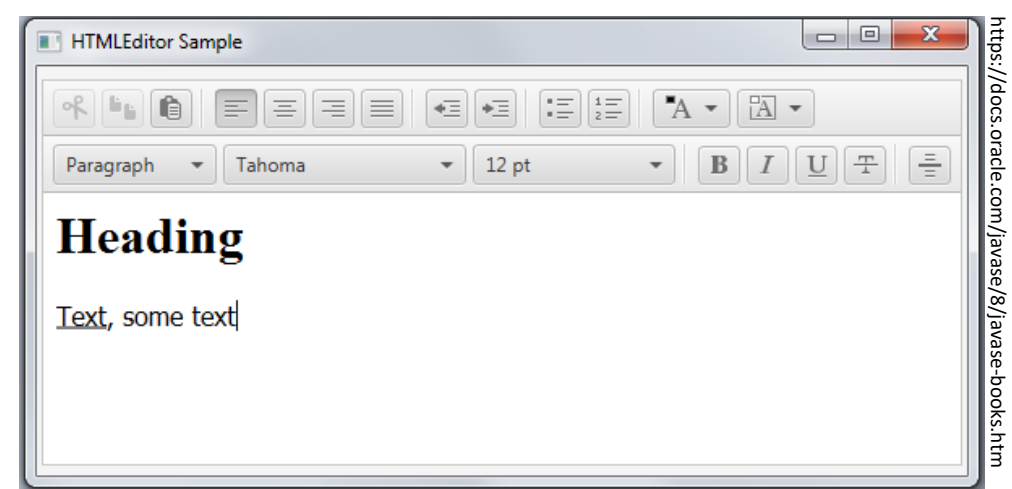
- Mehrzeilige Texte



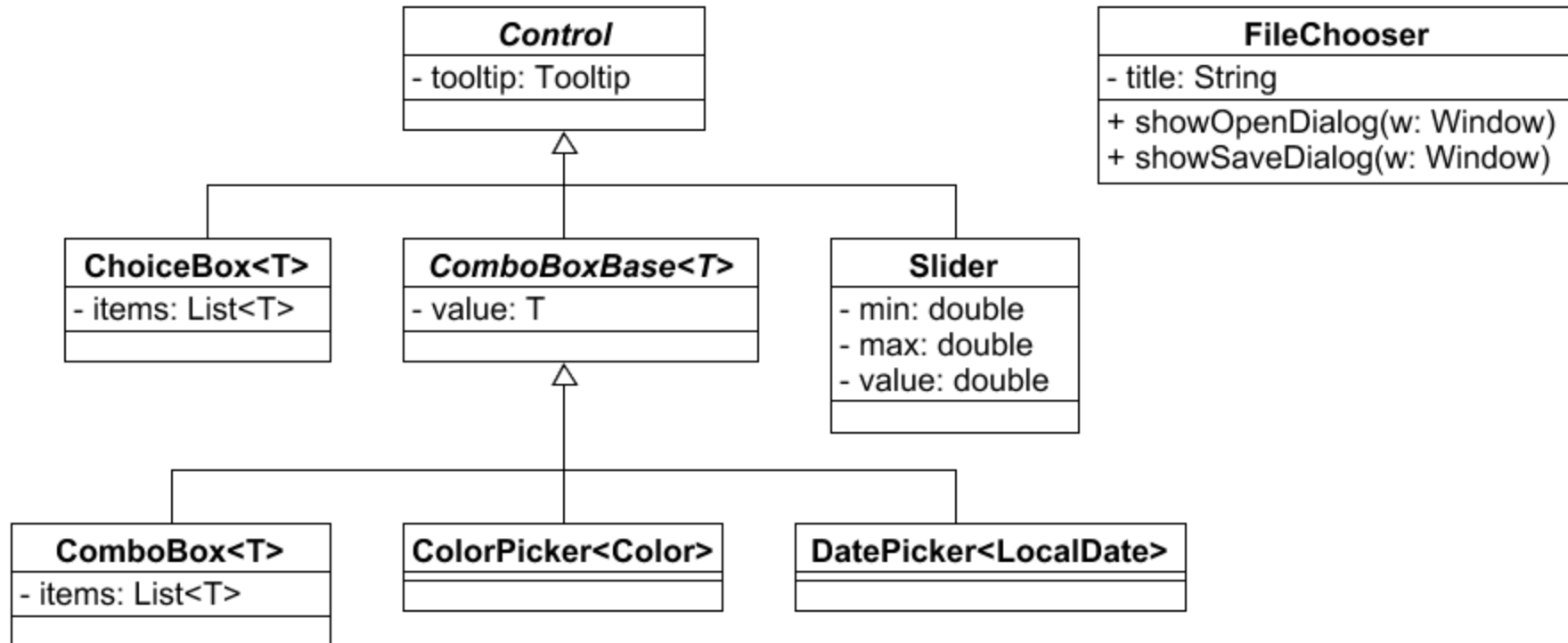
<https://docs.oracle.com/javase/8/javase-books.htm>

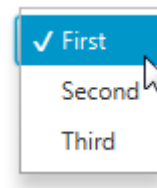
```
TextArea textArea = new TextArea();  
textArea.setPrefRowCount(10);  
textArea.setPrefColumnCount(20);  
textArea.setWrapText(true);
```

- Erstellung von HTML-formatiertem Text



```
HTMLEditor htmlEditor = new HTMLEditor();
Button getText = new Button("Get Text");
getText.setOnAction((event) ->
    System.out.println(htmlEditor.getHtmlText()));
VBox root = new VBox();
root.getChildren().addAll(htmlEditor, getText);
```



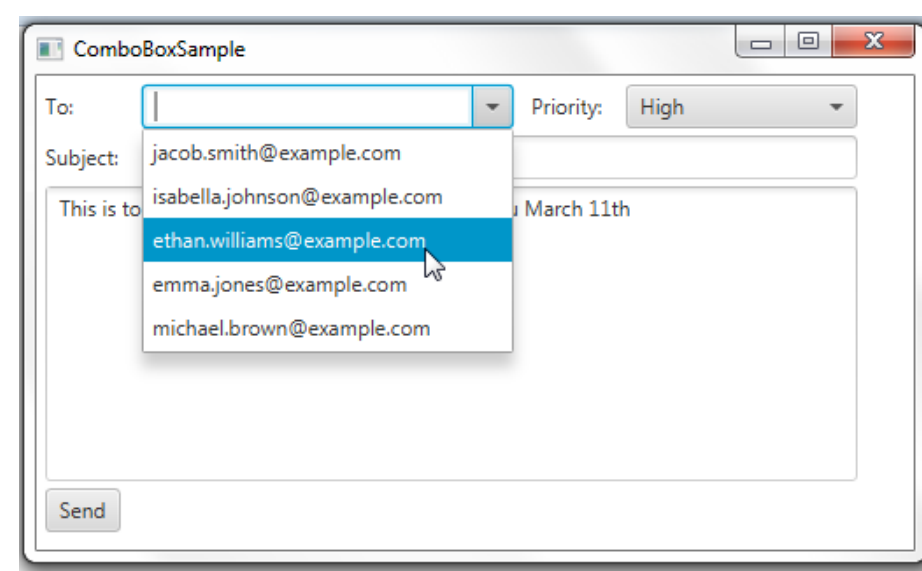


- Auswahl aus Liste von Optionen
- Keine Mehrfachauswahl

```
ChoiceBox<Person> cb = new ChoiceBox<Person>();
cb.getItems().add(new Person("Manuel", "Neuer", 28));
cb.getItems().add(new Person("Philipp", "Lahm", 30));
cb.getItems().add(new Person("Mats", "Hummels", 25));
cb.setConverter(new StringConverter<Person>() {
    @Override
    public String toString(Person p) {
        return p.getFirstName() + " " + p.getLastName();
    }
});
cb.getSelectionModel().selectedItemProperty().addListener(
    new ChangeListener<Person>() {
        @Override
        public void changed(ObservableValue<? extends Person> observable,
            Person oldValue, Person newValue) {
            System.out.println(newValue.getLastName());
        }
    }
);
```

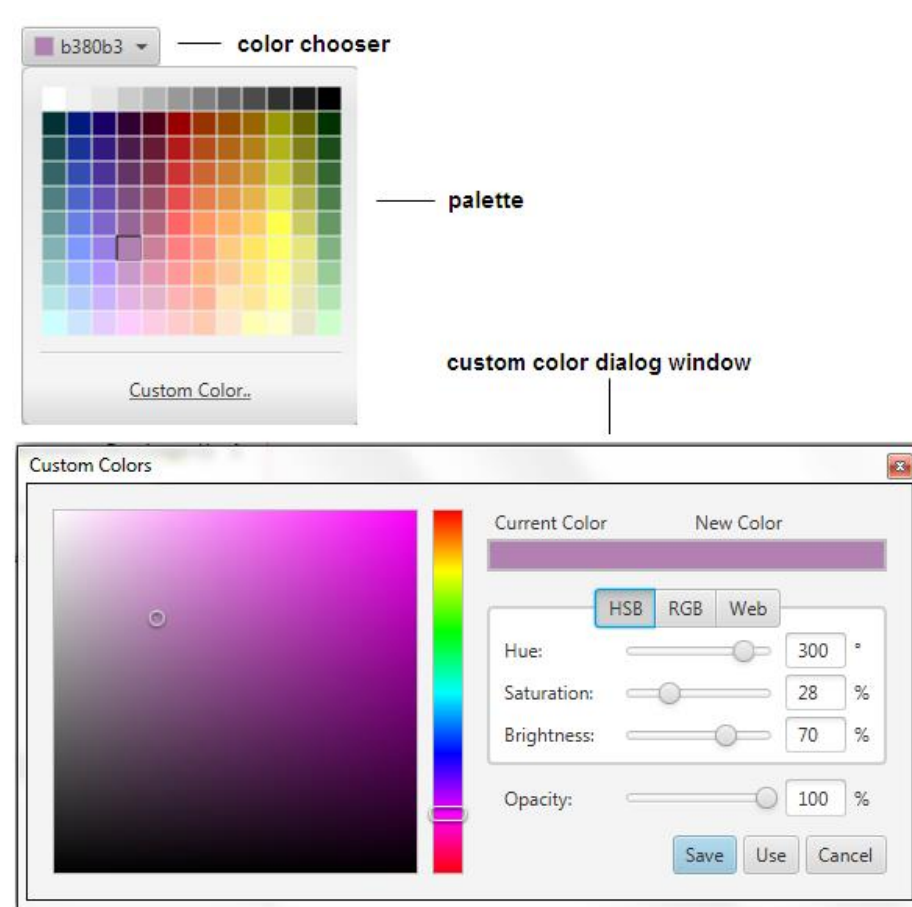
# ComboBox

- Ähnlich wie ChoiceBox
- Wesentlich mächtiger
- Auch editierbar



```
ComboBox<Person> cb = new ComboBox<>();  
cb.getItems().add(new Person("Manuel", "Neuer", 28));  
cb.getItems().add(new Person("Philipp", "Lahm", 30));  
cb.getItems().add(new Person("Mats", "Hummels", 25));  
cb.setPromptText("Bitte wählen");  
cb.setVisibleRowCount(2);  
cb.setEditable(true);  
// ...
```

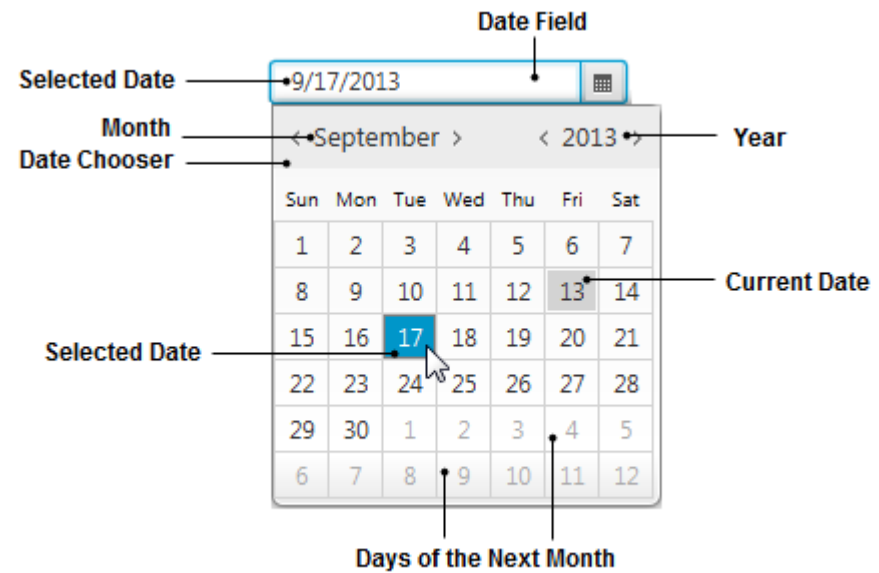
## ■ Farbauswahl



```
ColorPicker colorPicker = new ColorPicker();  
colorPicker.setOnAction((event) ->  
    System.out.println("Farbe " + colorPicker.getValue()));
```



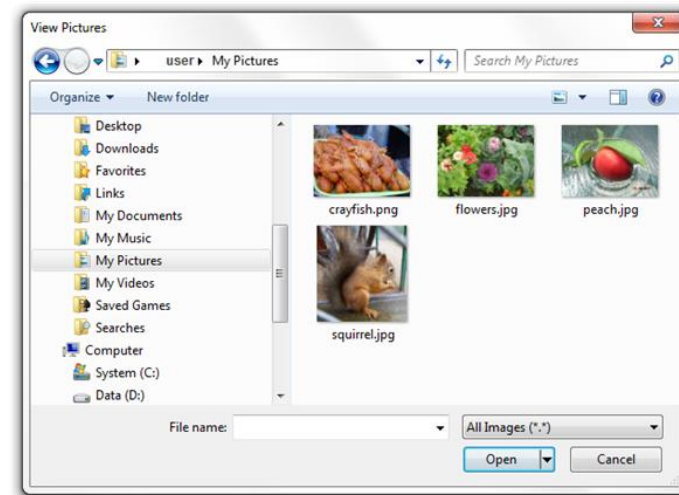
## ■ Datumsauswahl



```
DatePicker checkInDatePicker = new DatePicker();
checkInDatePicker.setOnAction((event) ->
    System.out.println("Datum " + checkInDatePicker.getValue()));
```

# FileChooser

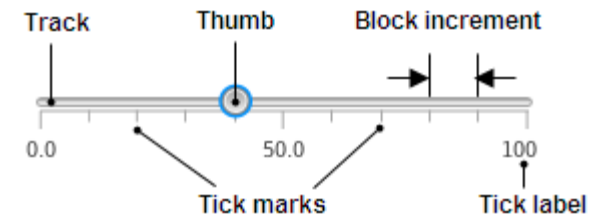
- Ermittlung von Dateien
- Eine oder mehrere



<https://docs.oracle.com/javase/8/javase-books.htm>

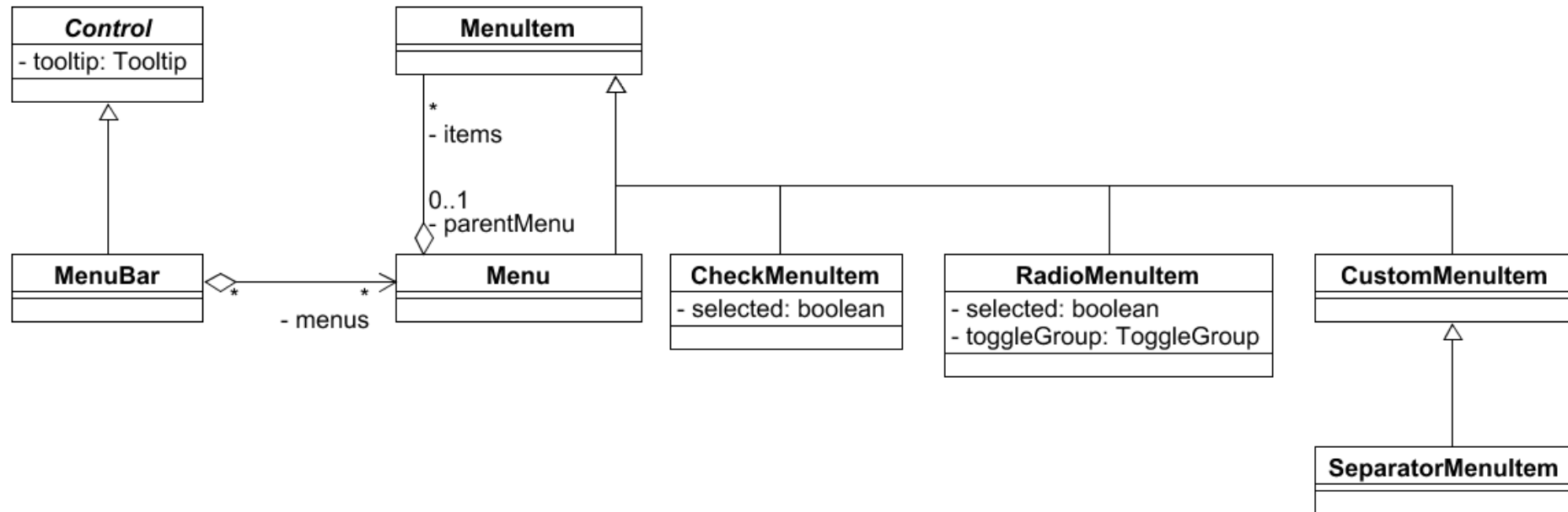
```
final FileChooser fileChooser = new FileChooser();  
File file = fileChooser.showOpenDialog(stage);  
if (file != null) { openFile(file); }  
  
List<File> list = fileChooser.showOpenMultipleDialog(stage);  
if (list != null) {  
    for (File file2 : list) {  
        openFile(file2);  
    }  
}
```

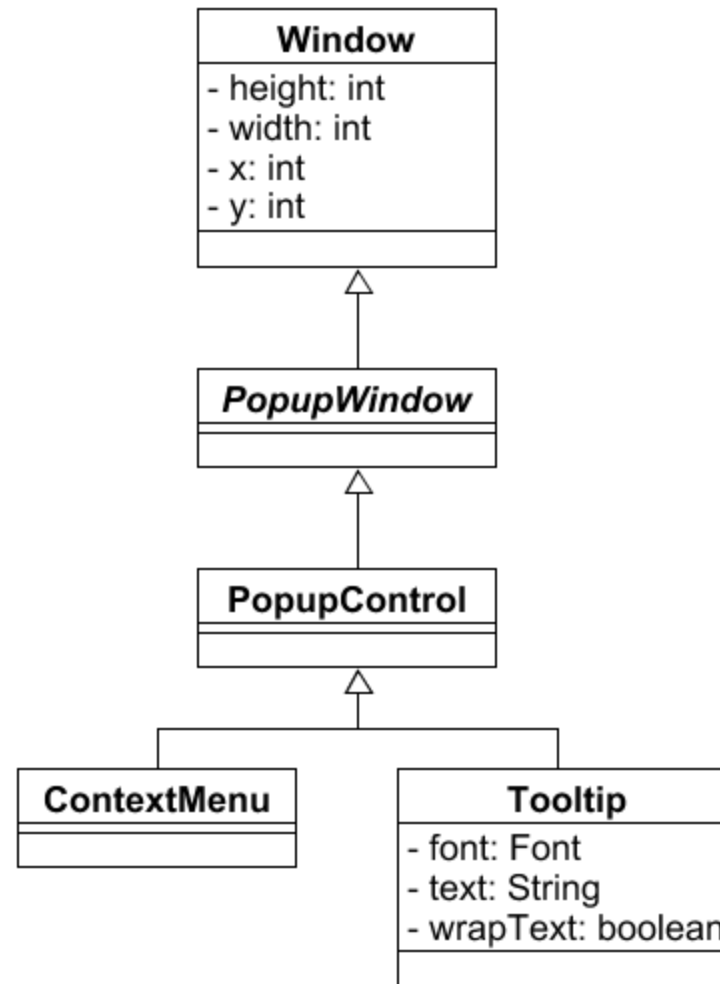
## ■ Auswahl numerischer Werte



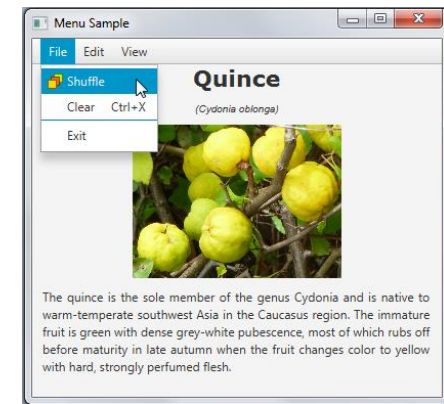
```
Slider slider = new Slider(0, 1, 0.5);
slider.valueChangingProperty().addListener(
    new ChangeListener<Boolean>() {
        @Override
        public void changed(ObservableValue<? extends Boolean> observable,
            Boolean oldValue, Boolean newValue) {
            circle.setScaleX(slider.getValue());
            circle.setScaleY(slider.getValue());
        }
    });
slider.setOrientation(Orientation.VERTICAL);
slider.setShowTickMarks(true);
slider.setShowTickLabels(true);
slider.setMajorTickUnit(.1);
slider.setMinorTickCount(4);
slider.setSnapToTicks(true);
```

# Menüs





- Menüleiste mit Menüs und Items
- Menüs und Untermenüs



<https://docs.oracle.com/javase/8/javase-books.htm>

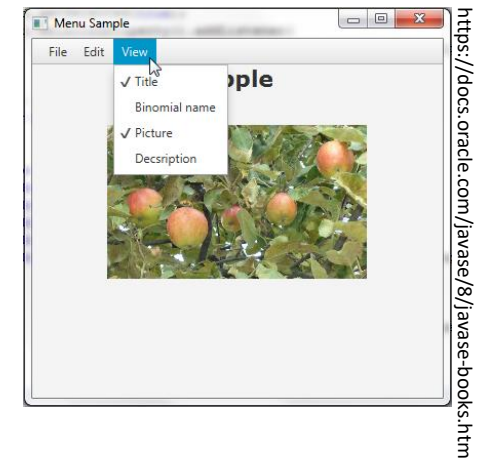
```
BorderPane root = new BorderPane();
MenuBar menuBar = new MenuBar();
Menu fileMenu = new Menu("Datei");
MenuItem openProjectMenuItem = new MenuItem("Projekt öffnen...");
MenuItem quitMenuItem = new MenuItem("Beenden");
quitMenuItem.setOnAction((event) -> Platform.exit());
Menu subMenu = new Menu("Untermenu");
subMenu.getItems().addAll(
    new MenuItem("Item 1"), new MenuItem("Item2"));
fileMenu.getItems().addAll(openProjectMenuItem, subMenu,
    new SeparatorMenuItem(), quitMenuItem);
Menu helpMenu = new Menu("Hilfe");
menuBar.getMenus().addAll(fileMenu, helpMenu);
root.setTop(menuBar);
```

- Einem Node zugeordnetes Menü

```
Label cLabel = new Label("Mit ContextMenu");
ContextMenu contextMenu = new ContextMenu();
MenuItem halloMenuItem = new MenuItem("Sag 'Hallo'");
halloMenuItem.setOnAction((event) ->
    System.out.println("Duke: 'Hallo!'"));
contextMenu.getItems().add(halloMenuItem);
cLabel.setOnContextMenuRequested((event) ->
    contextMenu.show(cLabel, event.getScreenX(), event.getScreenY()));
```



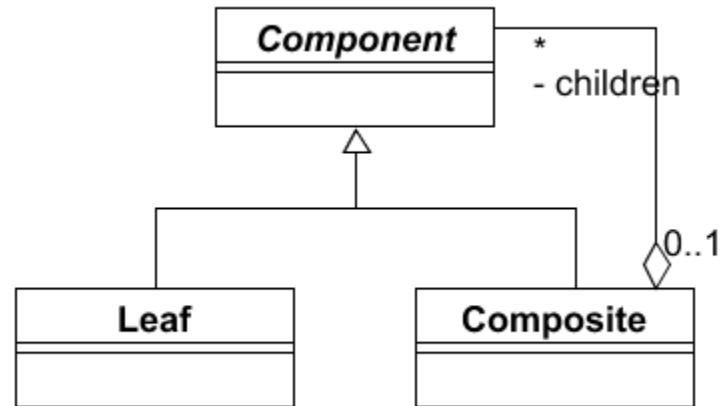
# MenuItems



```
CheckMenuItem showWorldMenuItem = new CheckMenuItem("anzeigen");
ToggleGroup toggleGroup = new ToggleGroup();
RadioMenuItem smallWorldMenuItem = new RadioMenuItem("Kleine Welt");
smallWorldMenuItem.setToggleGroup(toggleGroup);
fileMenu.getItems().add(smallWorldMenuItem);
RadioMenuItem largeWorldMenuItem = new RadioMenuItem("Große Welt");
largeWorldMenuItem.setToggleGroup(toggleGroup);
fileMenu.getItems().add(largeWorldMenuItem);
Slider s1 = new Slider(0, 1, 1);
CustomMenuItem customMenuItem = new CustomMenuItem(s1, false);
fileMenu.getItems().add(customMenuItem);
```

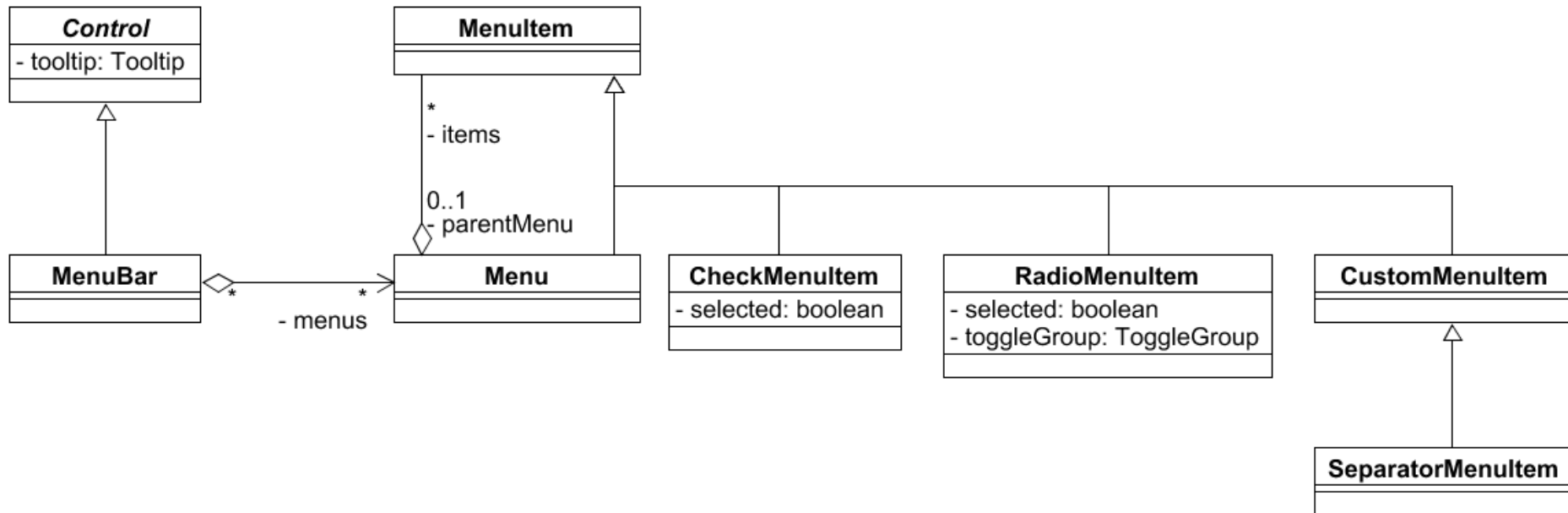


# Composite-Pattern



Setzt Objekte zu Baumstrukturen zusammen, die Teil-Ganzes Hierarchien repräsentieren.  
Einzelne Objekte und Kompositionen können auf die gleiche Weise behandelt werden.

# Composite-Pattern: Beispiel

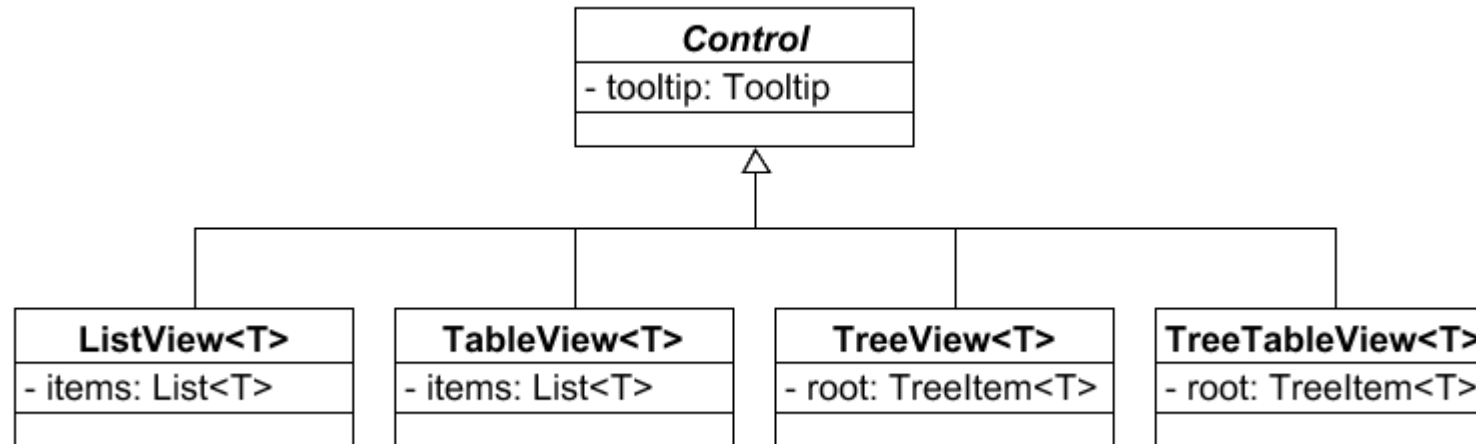


# MenuItems – Mnemonics

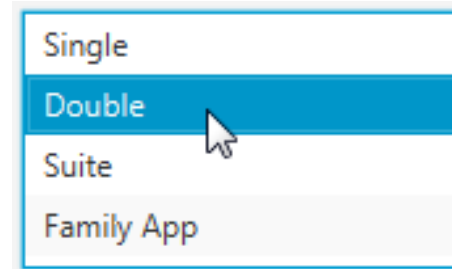
- Mnemonics
  - Präfix \_
  - werden ggf. mit ALT-Taste sichtbar
- Acceleratoren
  - Buchstabe + <Strg | Cmd>

```
Menu fileMenu = new Menu("_File");  
MenuItem newFileMenuItem = new MenuItem("_New...");  
newFileMenuItem.setAccelerator(  
    KeyCombination.keyCombination("SHORTCUT+N"));  
newFileMenuItem.setOnAction((event) ->  
    System.out.println("Something new, this way comes"));
```

# Darstellung von Daten

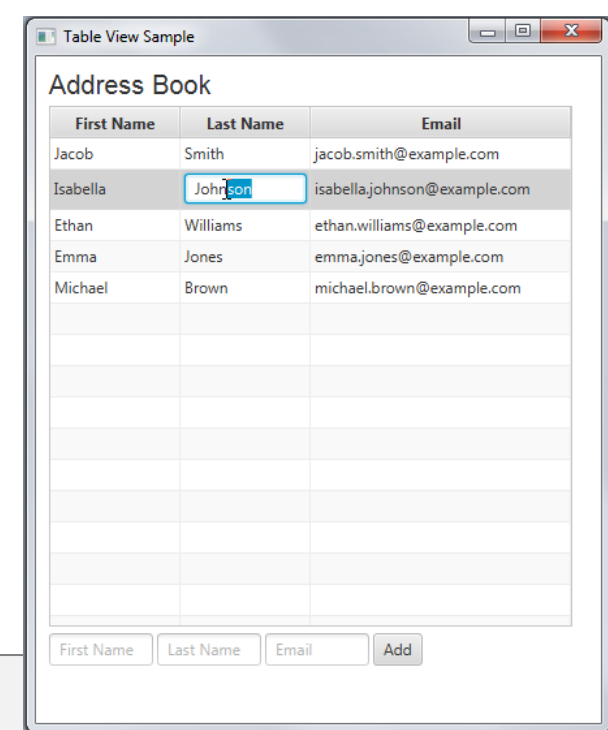


- Darstellung, Editieren und Auswahl von Daten
- Einzel- oder Mehrfachauswahl
- Darstellung von Zeilen → Cell-API



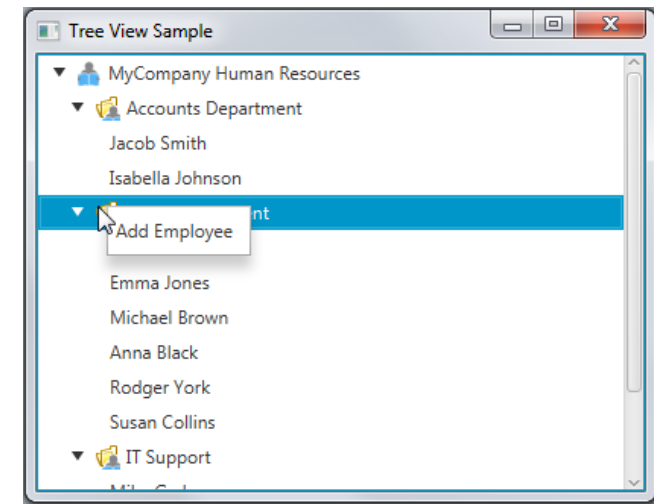
```
ListView<String> listView = new ListView<>();  
listView.getItems().addAll("Eins", "Zwei", "Drei", "Vier");
```

# TableView



```
TableView<Person> table = new TableView<>();
table.setEditable(true);
TableColumn firstNameCol = new TableColumn("First Name");
firstNameCol.setMinWidth(100);
firstNameCol.setCellValueFactory(
    new PropertyValueFactory<>("firstName"));
// ...
TableColumn emailCol = new TableColumn("Email");
emailCol.setMinWidth(200);
emailCol.setCellValueFactory(new PropertyValueFactory<>("email"));
table.setItems(data);
table.getColumns().addAll(firstNameCol, lastNameCol, emailCol);
```

- Darstellung von hierarchischen Daten



```
TreeItem<String> rootItem = new TreeItem<>("Inbox", rootIcon);
rootItem.setExpanded(true);
for (int i = 1; i < 6; i++) {
    TreeItem<String> item = new TreeItem<>("Message" + i);
    rootItem.getChildren().add(item);
}
TreeView<String> tree = new TreeView<>(rootItem);
```

# TreeTableView

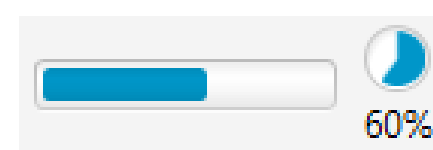
- Kombination von TreeView und TableView

Employee	Email
▼ Sales Department	
Ethan Williams	ethan.williams@example.com
Emma Jones	emma.jones@example.com
Michael Brown	michael.brown@example.com
Anna Black	anna.black@example.com
Rodger York	roger.york@example.com
Susan Collins	susan.collins@example.com

<https://docs.oracle.com/javase/8/javase-books.htm>



# ProgressIndicator/-Bar

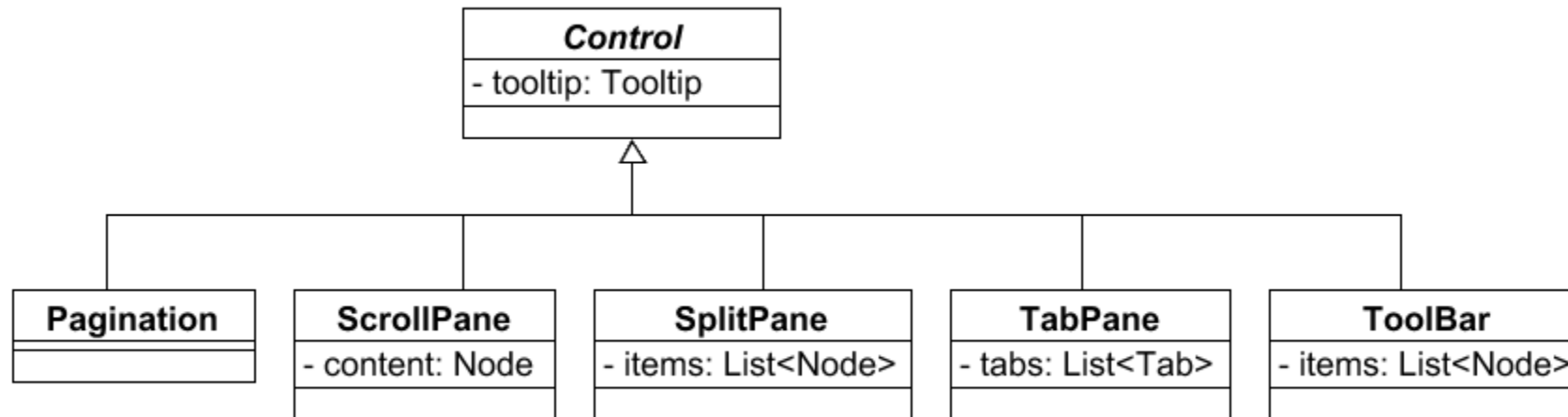


```
Slider slider = new Slider();
slider.setMin(0);
slider.setMax(50);

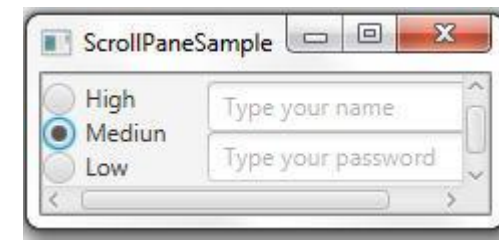
ProgressBar progressbar = new ProgressBar(0);
ProgressIndicator progressindicator = new ProgressIndicator(0);

slider.valueProperty().addListener(new ChangeListener<Number>() {
    @Override
    public void changed(ObservableValue<? extends Number> observable,
        Number oldValue, Number newValue) {
        progressbar.setProgress(newValue.doubleValue() / 50);
        progressindicator.setProgress(newValue.doubleValue() / 50);
    }
});
```

# Container



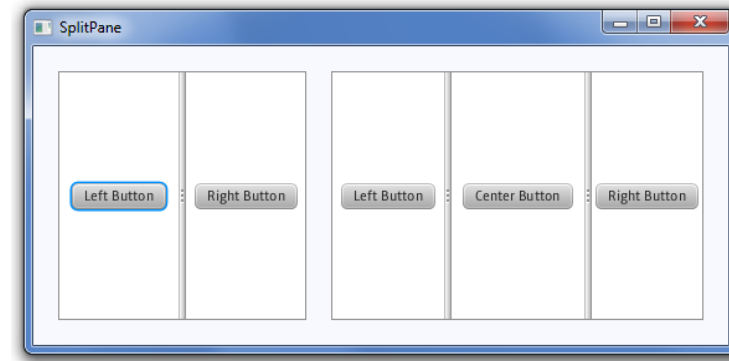
- Scrollt ein zugeordnetes Node-Objekt wenn nötig



<https://docs.oracle.com/javase/8/javase-books.htm>

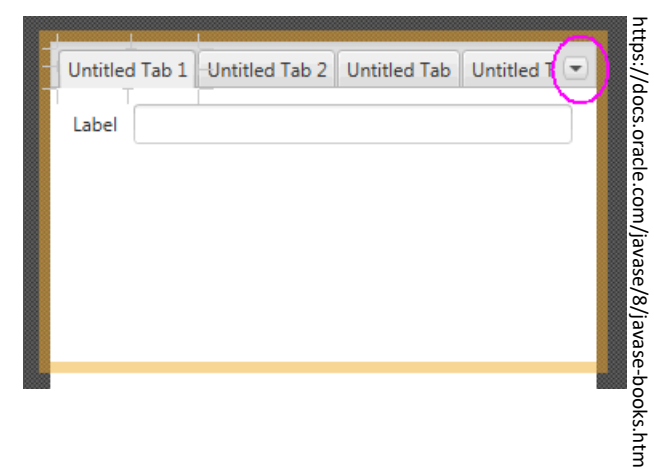
```
VBox vbox = new VBox();  
// ...  
ScrollPane scrollpane = new ScrollPane();  
scrollpane.setVmax(440);  
scrollpane.setPrefSize(115, 150);  
scrollpane.setContent(vbox);  
scrollpane.setHbarPolicy(ScrollBarPolicy.NEVER);  
scrollpane.setVbarPolicy(ScrollBarPolicy.AS_NEEDED);  
VBox.setVgrow(scrollpane, Priority.ALWAYS);
```

- Trennung von Node-Objekten durch Divider
- Divider können mit der Maus verschoben werden
- Aufnahme beliebig vieler Node-Objekte
- Horizontal oder vertikal



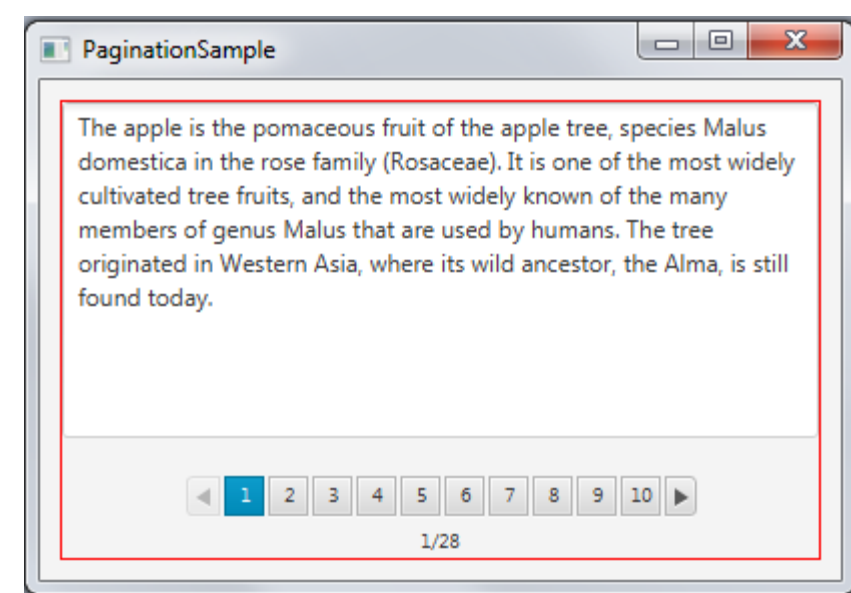
```
SplitPane splitPane = new SplitPane();  
for (int i = 0; i < 5; i++) {  
    splitPane.getItems().add(new StackPane(new Label("Pane" + i)));  
}
```

- Verwaltung von Tab-Objekten (Reiter)
  - versehen ein Node-Objekt mit einem Tab
  - Text + Grafik
- Ein Tab-Node ist sichtbar
- Ggf. Overflow-Bereich



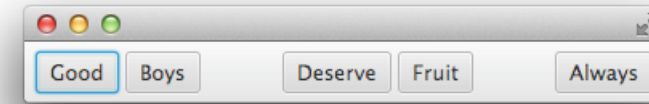
```
TabPane tabPane = new TabPane();
for (int i = 0; i < 20; i++) {
    Tab tab = new Tab("JavaFXDemo" + i + ".java");
    tab.setGraphic(new ImageView(new Image("java.png")));
    Text text = new Text("Dies ist der Inhalt von Dokument " + i);
    tab.setContent(new StackPane(text));
    tabPane.getTabs().add(tab);
}
```

- Navigation durch mehrere Seiten



<https://docs.oracle.com/javase/8/javase-books.htm>

```
Pagination pagination = new Pagination(28, 0);
pagination.setPageFactory(new Callback<Integer, Node>() {
    @Override
    public Node call(Integer i) {
        return new StackPane(new ImageView(
            new Image("img" + i + ".png")));
    }
});
```



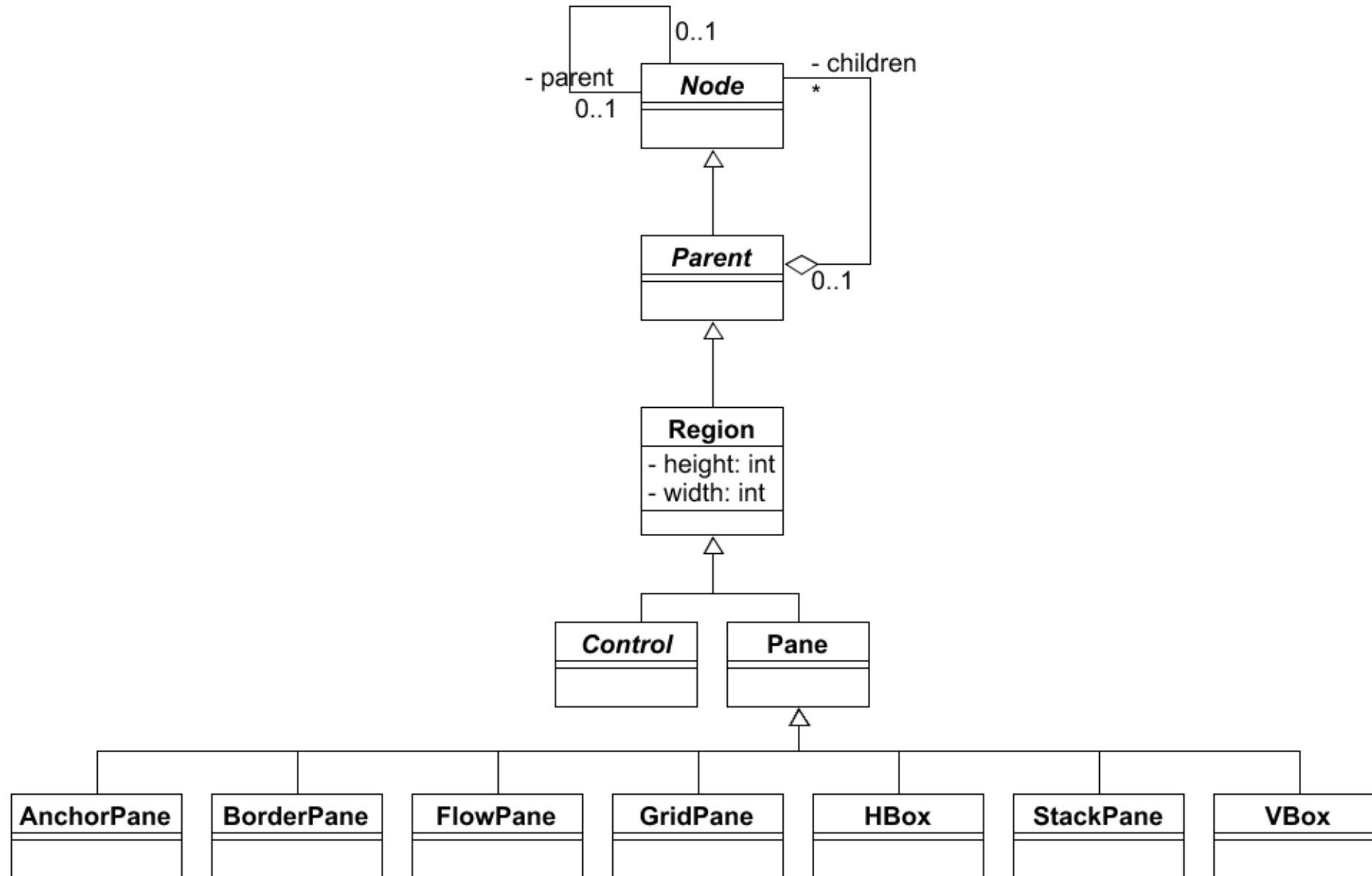
- Container für Node-Objekte (meistens Buttons)
- Horizontale oder vertikale Ausrichtung
- Wenn nicht alle Objekte hineinpassen, erscheint ein Overflow-Button

```
BorderPane contentPane = new BorderPane();
ToolBar toolBar = new ToolBar(new Button("New"),
                             new Button("Open"), new Button("Save"),
                             new Separator(), new Button("Clean"),
                             new Button("Compile"), new Button("Run"),
                             new Separator(), new Button("Debug"),
                             new Slider());
contentPane.setTop(toolBar);
```

# Layouts



- Positionierung von **Node**-Objekten innerhalb von Panes
- Absolut möglich (→ Klasse **Pane**)
- Layout-Panes: Verteilung des Platzes auf alle Nodes eines Panes
- Unterschiedliche Verteilungsstrategien
- Constraints:
  - Beschränkung von Position und Größe von **Node**-Objekten
  - Werden über statische Methoden der **Layout**-Klassen gesetzt; diese setzen im **Node**-Objekt eine **LayoutProperty**
- Schachtelung von Layout-Objekten möglich  
→ Komplexe Layouts!
- Ausschalten von Layout-Constraints für ein **Node**-Objekt via **node.setManaged(false);**



- Für absolute Positionierung geeignet

```
Pane canvas = new Pane();
canvas.setPrefSize(200, 200);
Circle circle = new Circle(50, Color.BLUE);
circle.relocate(20, 20);
Rectangle rectangle = new Rectangle(100, 100, Color.RED);
rectangle.relocate(70, 70);
Button button = new Button("Klick mich");
button.relocate(90, 110);
canvas.getChildren().addAll(circle, rectangle, button);
Scene scene = new Scene(canvas, 300, 250);
```

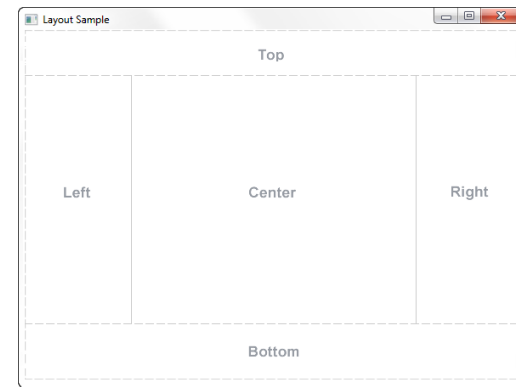
- Kind-Komponenten werden neben-/untereinander gelegt



```
VBox vbox = new VBox();  
vbox.setPadding(new Insets(10, 20, 30, 40));  
vbox.setSpacing(50);  
Button claimExcessSpace = new Button("lass mich wachsen!");  
claimExcessSpace.setMaxHeight(300);  
VBox.setVgrow(claimExcessSpace, Priority.ALWAYS);  
Button staySmall = new Button("lass mich (zunaechst) in Ruhe!");  
staySmall.setPrefSize(200, 200);  
staySmall.setMaxHeight(1000);  
VBox.setVgrow(staySmall, Priority.SOMETIMES);  
vbox.getChildren().addAll(claimExcessSpace, staySmall);
```

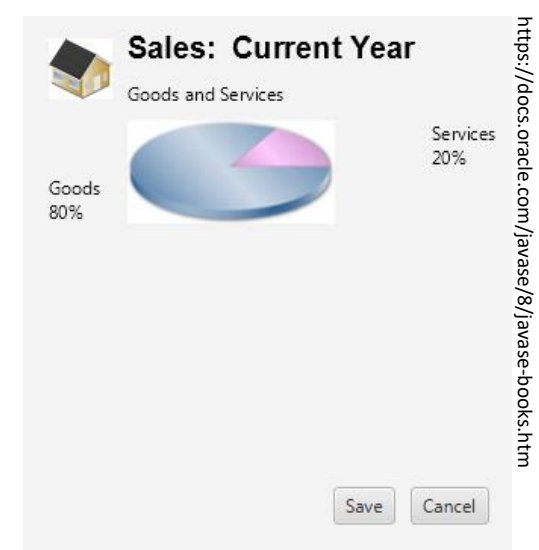
# BorderPane

- Bis zu fünf Komponenten
  - top, bottom, left, right, center



```
BorderPane borderPane = new BorderPane();
Label toolbar = new Label("toolbar");
borderPane.setTop(toolbar);
Label taskbar = new Label("taskbar");
borderPane.setBottom(taskbar);
Label document = new Label("document");
borderPane.setCenter(document);
Label navigator = new Label("navigator");
borderPane.setLeft(navigator);
Label properties = new Label("properties");
borderPane.setRight(properties);
BorderPane.setAlignment(toolbar, Pos.CENTER);
primaryStage.setScene(new Scene(borderPane, 300, 200));
```

- Positionierung von Komponenten in bestimmtem Abstand vom Rand



```
AnchorPane anchorPane = new AnchorPane();
Button save = new Button("save");
Button help = new Button("help");
Button cancel = new Button("cancel");
HBox buttons = new HBox();
buttons.setSpacing(12);
buttons.getChildren().addAll(cancel, save, help);
anchorPane.getChildren().add(buttons);
AnchorPane.setRightAnchor(buttons, 10.0);
AnchorPane.setBottomAnchor(buttons, 20.0);
primaryStage.setScene(new Scene(anchorPane, 300, 200));
```

- Solange Platz da ist, werden Komponenten nebeneinander gelegt
- Ansonsten wird umgebrochen
- Horizontale oder vertikale Anordnung



<https://docs.oracle.com/javase/8/javase-books.htm>



```
FlowPane iconView = new FlowPane();
iconView.setVgap(10);
iconView.setHgap(20);
for (int i = 0; i < images.length; i++) {
    iconView.getChildren().add(new ImageView(images[i]));
}
primaryStage.setScene(new Scene(iconView, 300, 200));
```

- Übereinanderplatzierung der Komponenten (z-Koordinate)

```
Button ok = new Button("OK");  
Button cancel = new Button("Cancel");  
HBox hBox = new HBox(cancel, ok);  
Button button = new Button("Do Something!");  
StackPane root = new StackPane(button, hBox);  
primaryStage.setScene(new Scene(root, 300, 200));
```

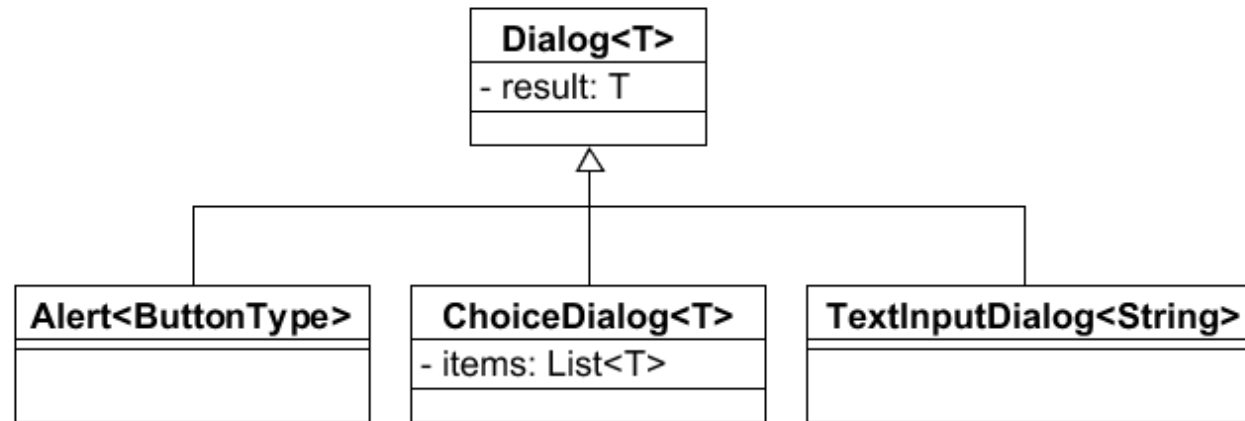
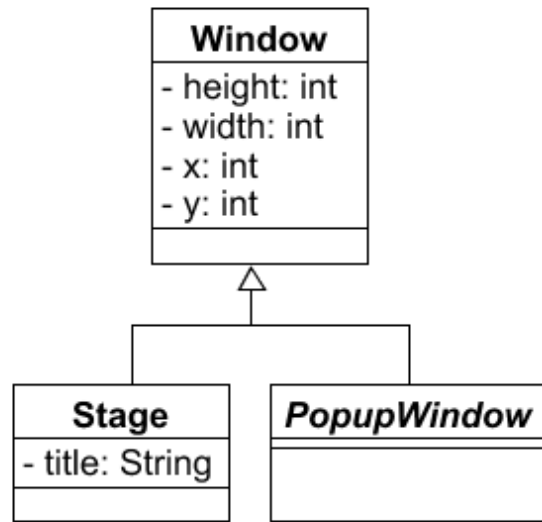


- Layout basiert auf Gitter
- Definition komplexer Abhängigkeiten ist möglich
- Platzierung von Komponenten durch Angabe von Reihe, Spalte und ggf. Row-Span und Col-Span
- Anpassung von Höhe und Breite mittels ColumnConstraints und RowConstraints

	<b>Sales: Current Year</b>		
	Goods and Services		
			Services 20%
Goods 80%			

<https://docs.oracle.com/javase/8/javase-books.htm>

# Fenster



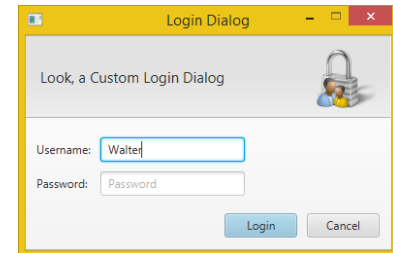
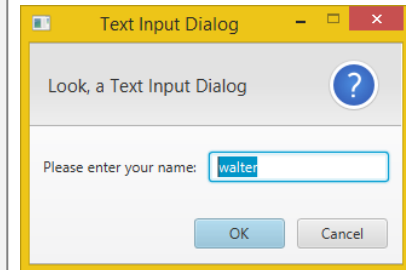
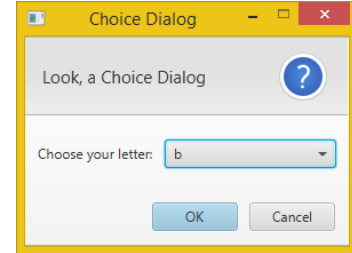
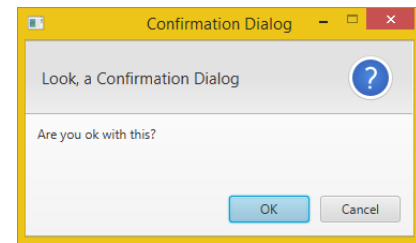
- Fenster = Stage
- Eine intern erzeugte Stage wird der **Application.start** Methode übergeben
- Weitere eigene Stages sind möglich



```
Stage secondStage = new Stage();
secondStage.setTitle("Fenster 2");
secondStage.setScene(new Scene(new StackPane(
    new Label("Hello 2")), 200, 400));
secondStage.setX(40);
secondStage.setY(200);
secondStage.sizeToScene();
secondStage.show();
```

# Dialoge

```
private void showAlert() {  
    Alert alert = new Alert(AlertType.CONFIRMATION,  
        "Alles klar?", ButtonType.YES,  
        ButtonType.NO, ButtonType.CANCEL);  
    alert.showAndWait();  
    if (alert.getResult() == ButtonType.YES) {  
        // do stuff  
    }  
}
```



<https://docs.oracle.com/javase/8/javase-books.htm>

- JavaFX
  - Controls
  - Layout
  - Fenster