Object Oriented Programming 2

Topic 1 – JavaFX

Andres Scheidegger
(based on course GUI CAS SD-HS15 by Philipp Locher)

Berner Fachhochschule | Haute école spécialisée bernoise | Bern University of Applied Sciences

Goals

- You know the basic concepts of JavaFX (scene graph, nodes, controls, layout, events)
- You can program a GUI with Java and JavaFX (without FXML)
- You are able to handle events in your JavaFX GUI applications

Outline

- Introduction
- Scene Graph
- Controls
- Layout Panes
- Application
- Events

Graphische Benutzeroberflächen in Java

Überischt

Java bietet drei Technologien an, um graphische Benutzeroberflächen (GUI) zu implementieren:

- AWT - Abstract Windowing Toolkit (Java 1.1)
- Swing - Lightweight UI (Java 1.2)
- JavaFX (Java 8)

AWT - Abstract Windowing Toolkit...

- Ist ein plattform-unabhängiges GUI-Toolkit
  - Linux/Unix
  - Windows
  - OS X
- Ist plattform-spezifisch implementiert
  - Plattform-abhängiges Look & Feel
  - Bietet nicht alle Möglichkeiten der Plattform
- Enthält drei Kategorien von Klassen
  - User Interface Komponenten
  - Grafik Hilfsklassen
  - Layout Managers

Swing

AWT	$\Longleftrightarrow$	**Swing**
Heavyweight	$\Longleftrightarrow$	Lightweight

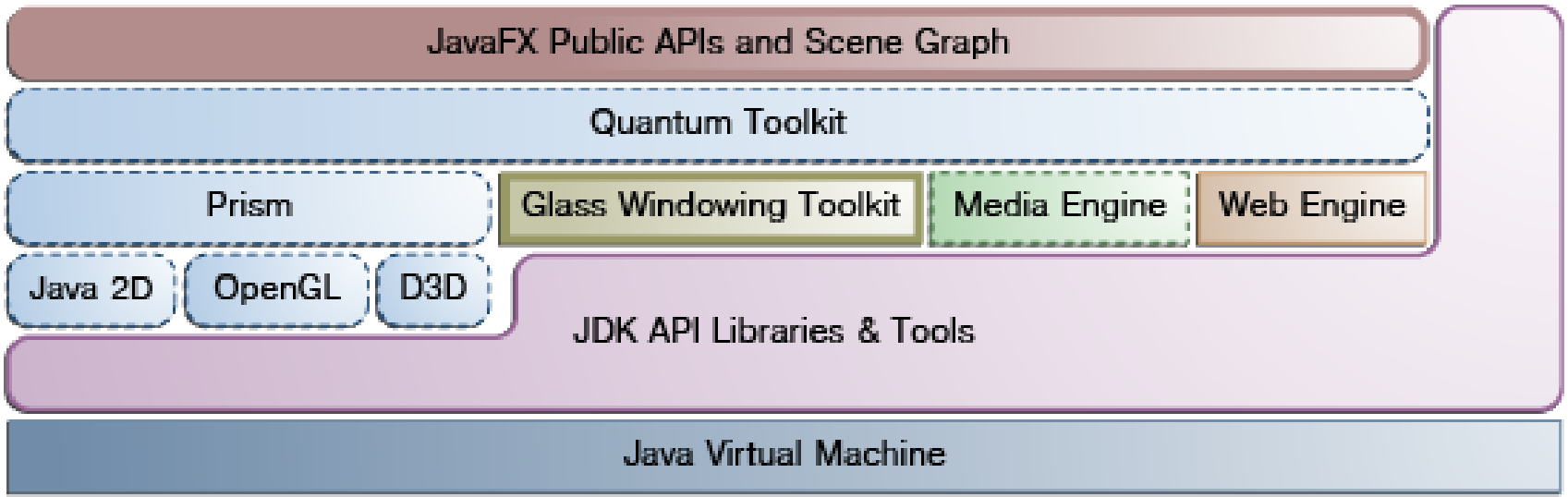
Swing basiert nicht wie AWT auf den nativen dem Betriebssystem zu Grunde liegenden GUI-Komponenten, sondern zeichnet (rendert) alle Komponenten selber.

Die Antwort von Java auf HTML5!

- FXML/CSS
- Scene Builder Tool
- Transformationen und Animationen
- 2D/3D
- Support für Audio/Video
- Charts package
- Properties und Bindings
- Multitouch support
- ...

(Version 1.X hat wenig bis nichts mit den Versionen ≥ 2 zu tun)

Die Architektur von JavaFX

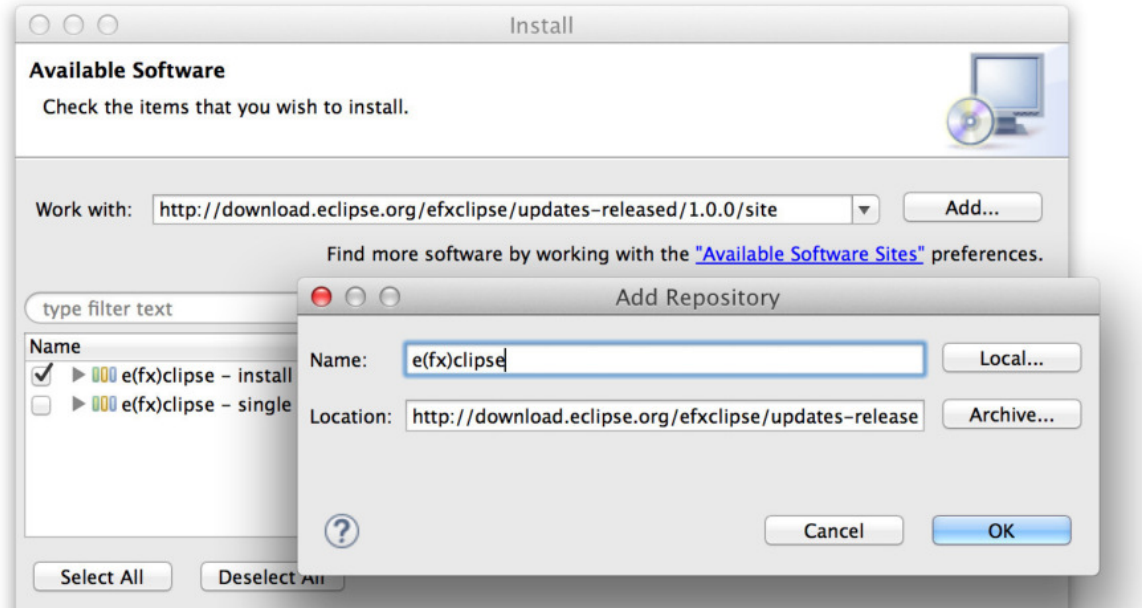


~~http://docs.oracle.com/javase/8/javafx~~

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

JavaFX mit Eclipse

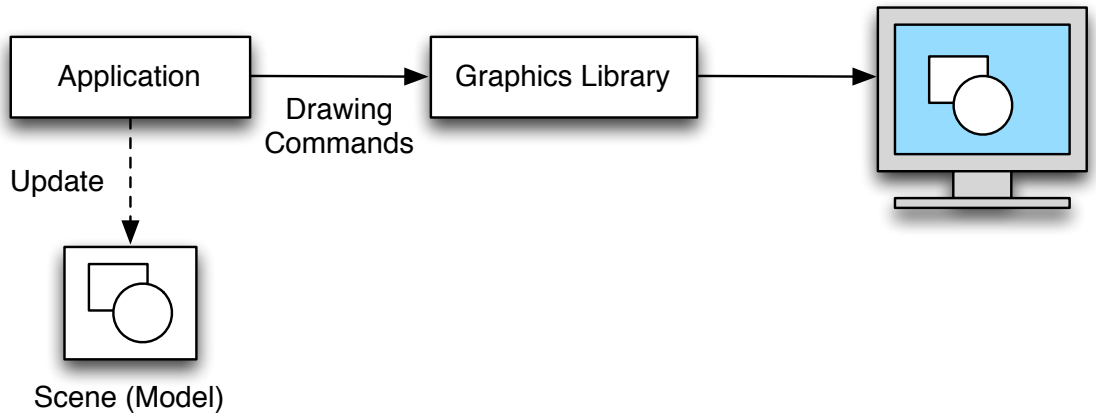
- JavaFX ist seit Java8 Bestandteil von JRE/JDK
- Die Unterstützt von JavaFX in Eclipse ist jedoch schwach. Es empfiehlt sich daher, zusätzlich **e(fx)clipse** zu installieren (http://www.eclipse.org/efxclipse)



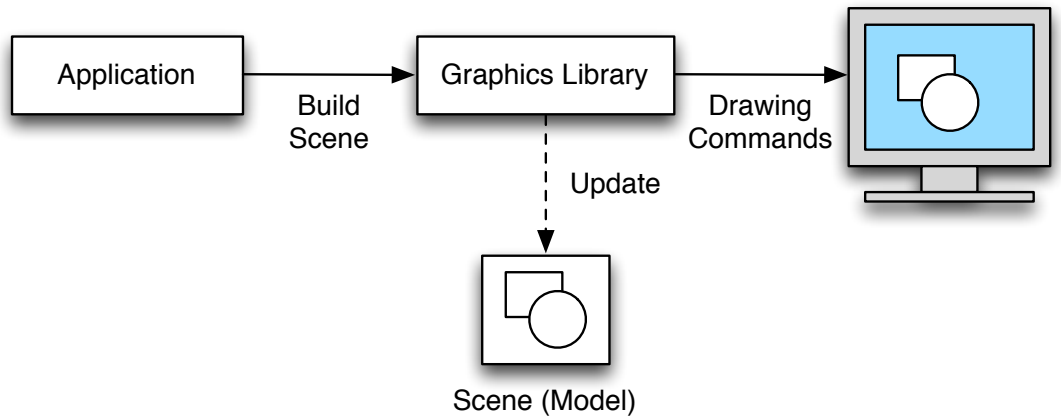
- Scene Builder muss extra installiert werden

Scene Graph

Graphik Modus

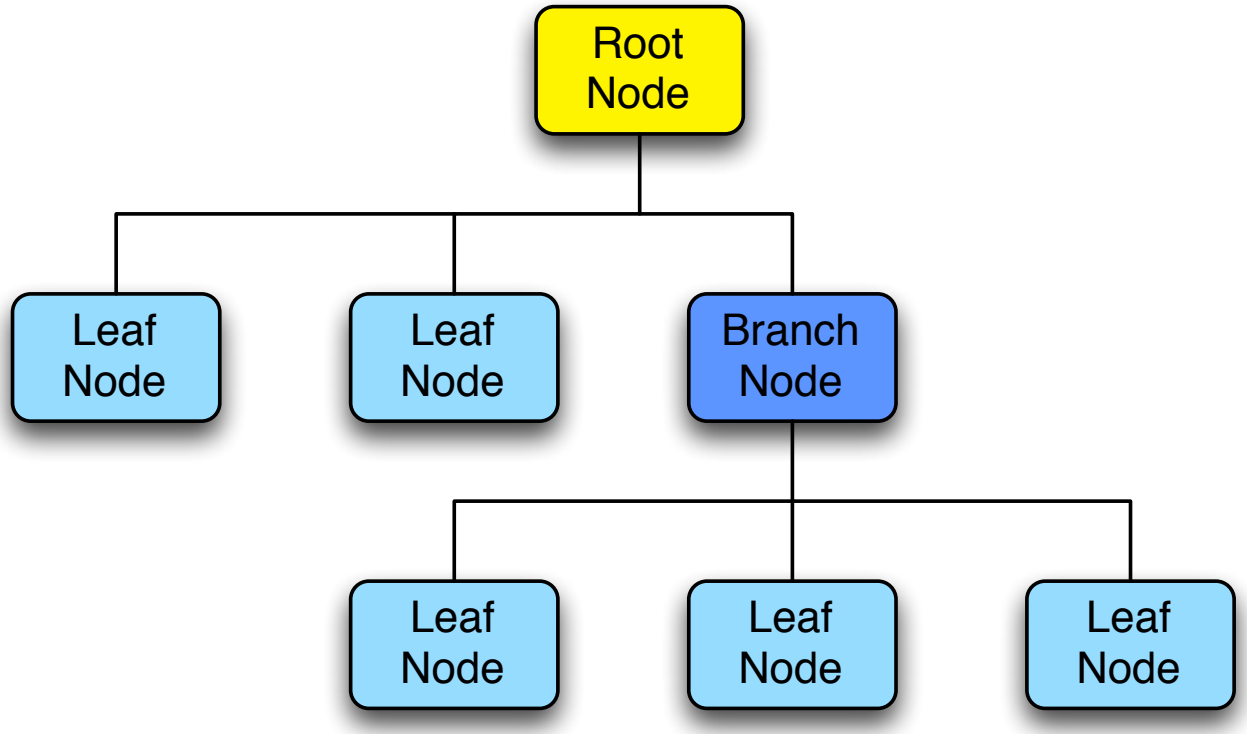


Immediate-Mode



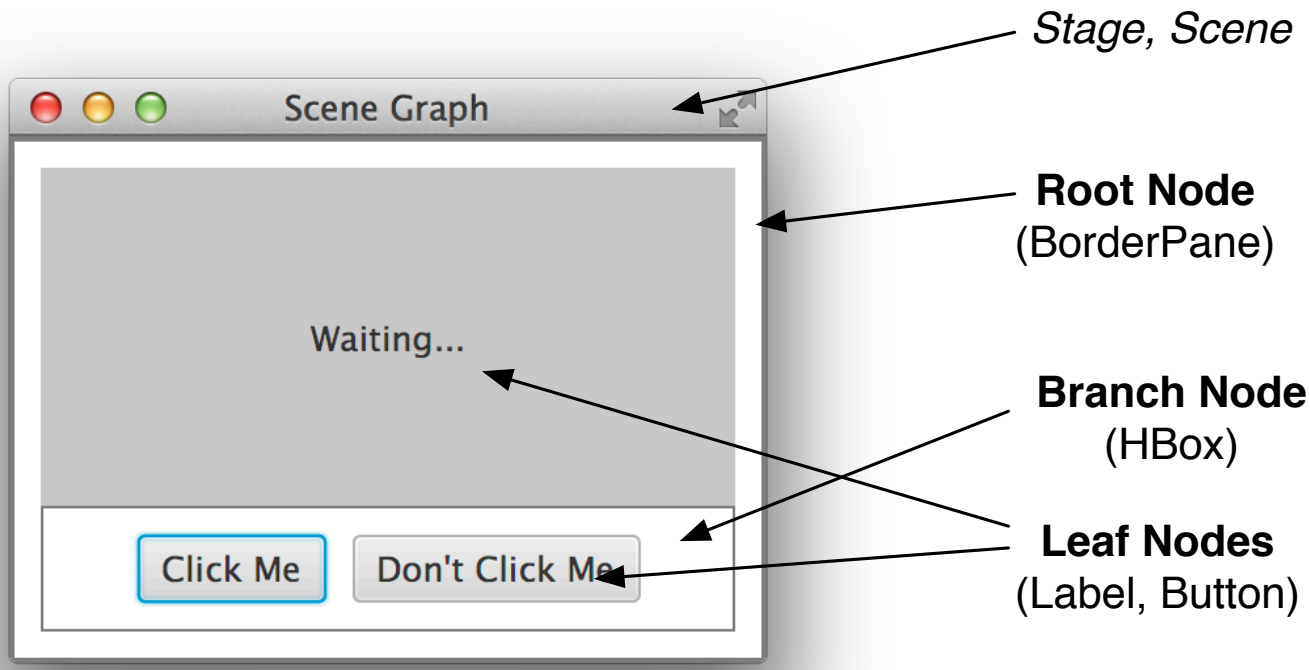
Retained-Mode

Scene Graph



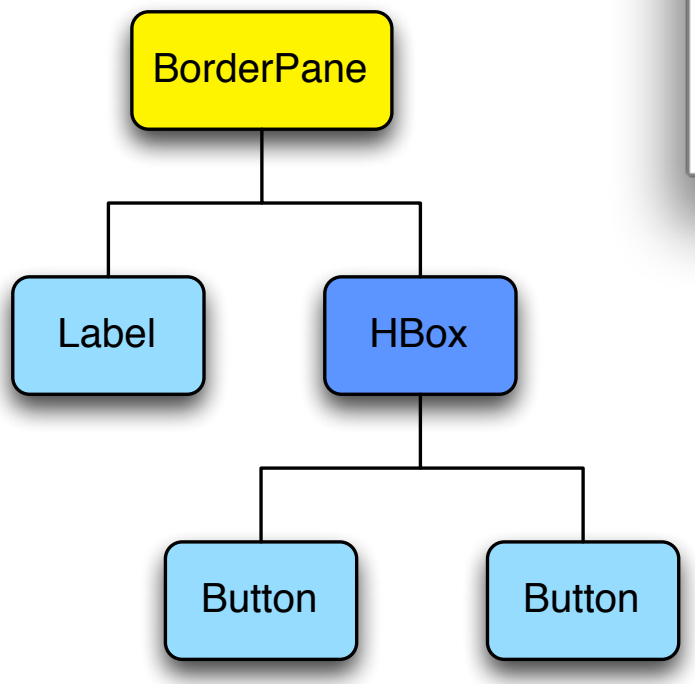
- Baumstruktur
- Jeder **Node** (ausser Root) hat genau einen **Parent** und keine oder mehre **Children**

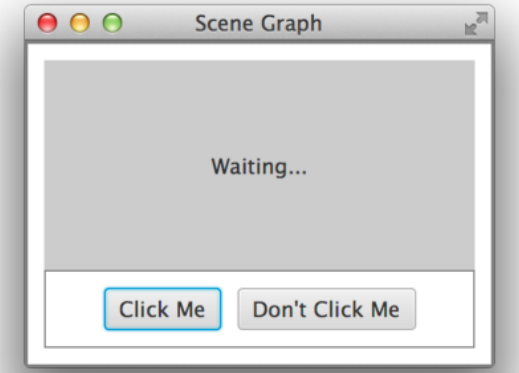
Scene Graph



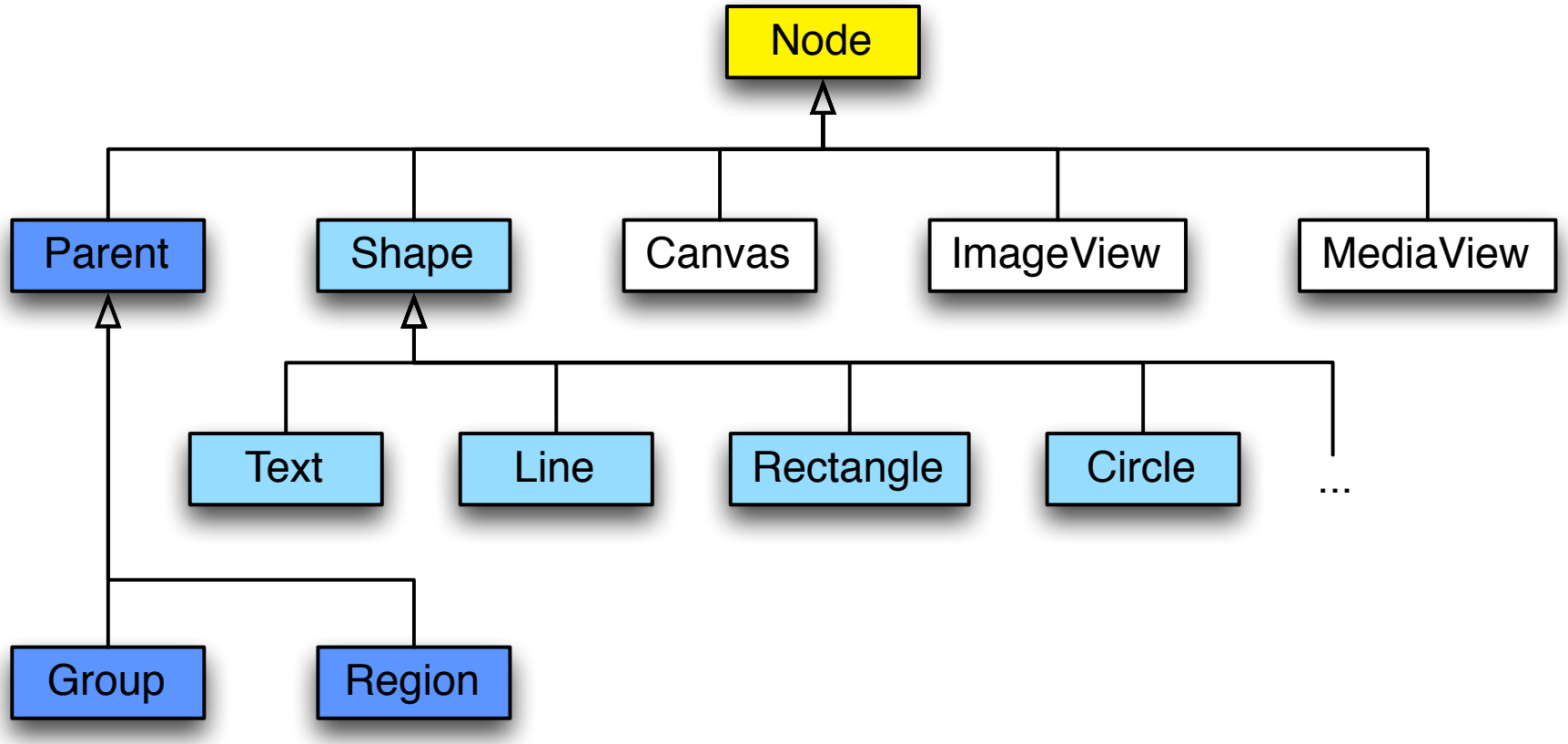
(Controls (Label, Button, . . .) sind in Wirklichkeit keine Leaf Nodes, können aber aus Sicht eines Entwicklers/Designers als solche betrachtet werden)

Scene Graph

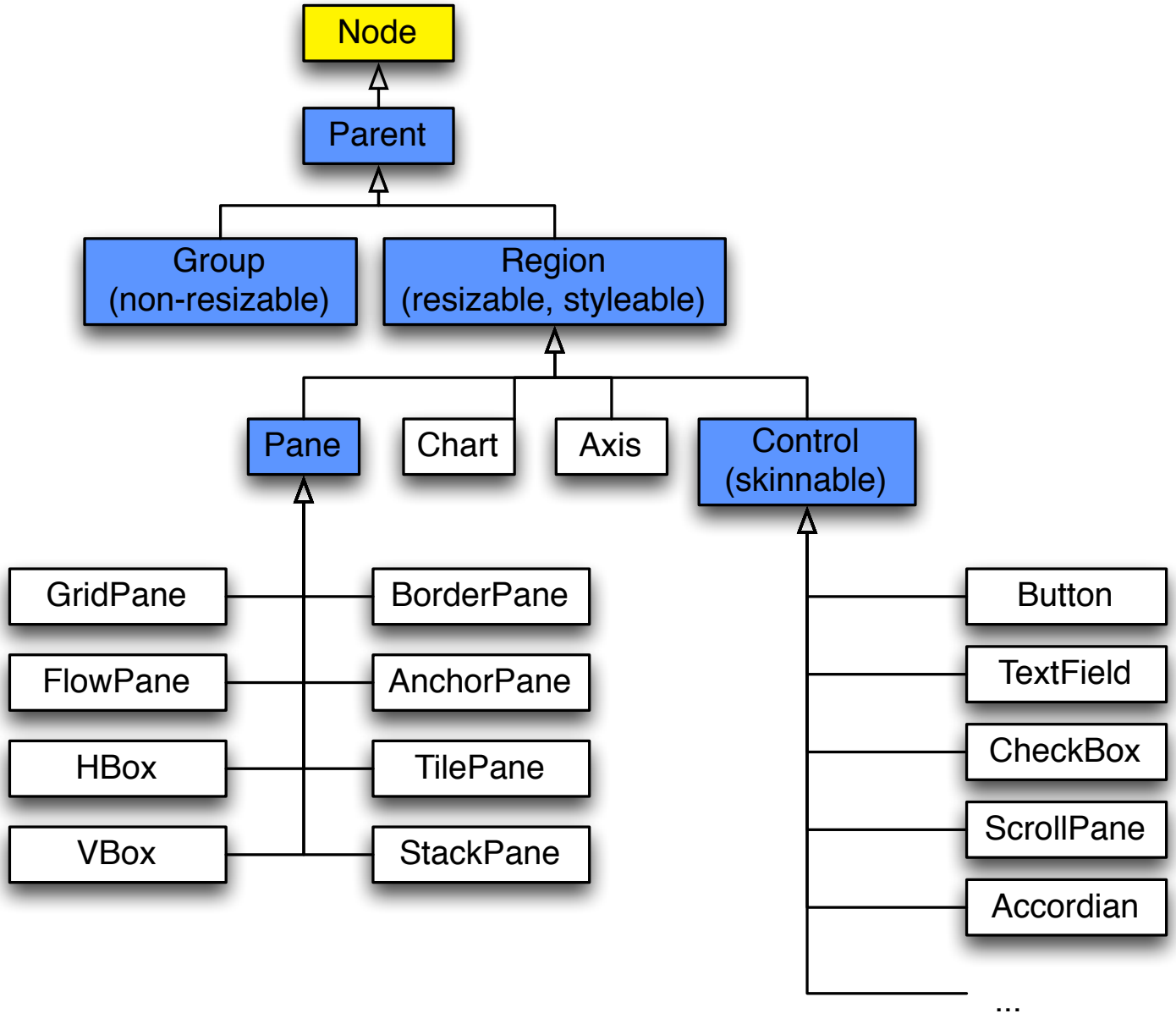

BorderPane
Label
HBox
Button
Button


Scene Graph
Waiting...
Click Me
Don't Click Me

Node

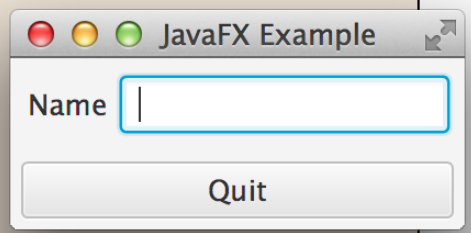

Node
Parent
Shape
Canvas
ImageView
MediaView
Text
Line
Rectangle
Circle
...
Group
Region

Branch Nodes



Example

```
public class JavaFXExample extends Application {
    @Override
    public void start(Stage stage) throws Exception {
        BorderPane root = new BorderPane();
        HBox hbox = new HBox(5); root.setCenter(hbox);

        // Create a label and a text field and add them to the scene graph
        Label l = new Label("Name");
        TextField t = new TextField();
        hbox.getChildren().addAll(l, t);

        // Create a button and add it to the scene graph
        Button b = new Button("Quit"); root.setBottom(b);

        // Set up the stage
        stage.setTitle("JavaFX Example");
        stage.setScene(new Scene(root, 200, 80));
        stage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```



JavaFXExample.java

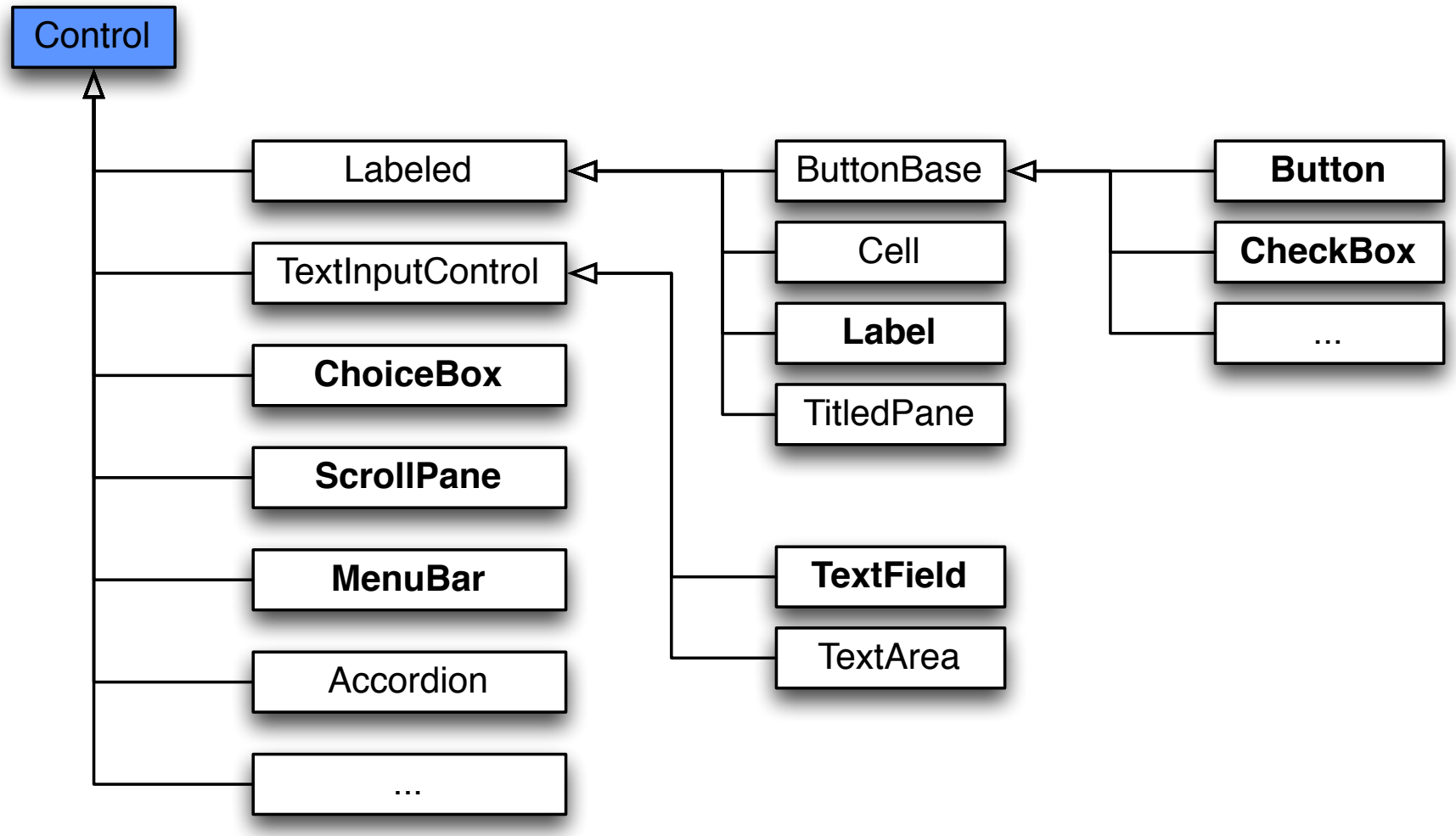
Controls

Controls

http://docs.oracle.com/javafx

https://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui_controls.htm

Controls

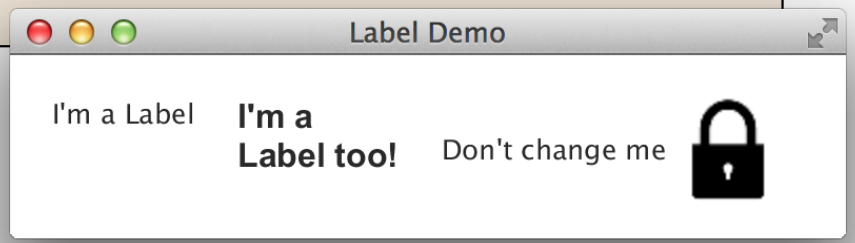

Control
Labeled
TextInputControl
ChoiceBox
ScrollPane
MenuBar
Accordion
...
ButtonBase
Cell
Label
TitledPane
TextField
TextArea
Button
CheckBox
...

Label

Ein `Label` kann Text und Bild enthalten

```
final Label label1 = new Label("I'm a Label");
root.getChildren().add(label1);

final Label label2 = new Label("I'm a\nLabel too!");
label2.setFont(Font.font("Dialog", FontWeight.BOLD, 16));
root.getChildren().add(label2);

final ImageView image = new ImageView(new Image("file:res/lock.png"));
final Label label3 = new Label("Don't change me", image);
label3.setContentDisplay(ContentDisplay.RIGHT);
root.getChildren().add(label3);
```

LabelExample.java



Button

- Ermöglicht dem Benutzer, bewusst Events auszulösen
- Text und/oder Icon zur Beschreibung der Aktion
- Mit Check-Boxen und Radio-Buttons werden Optionen ausgewählt
- Radio-Buttons werden in Toggle-Groups zusammengefasst

Button

```
final Button button = new Button("Click me");
root.getChildren().add(button);

final CheckBox cb1 = new CheckBox("Apple");
root.getChildren().add(cb1);
final CheckBox cb2 = new CheckBox("Pear");
root.getChildren().add(cb2);
// ...

final ToggleGroup tg = new ToggleGroup();
final RadioButton rb1 = new RadioButton("Red");
rb1.setToggleGroup(tg);
root.getChildren().add(rb1);
final RadioButton rb2 = new RadioButton("Blue");
rb2.setToggleGroup(tg);
root.getChildren().add(rb2);
// ...
```

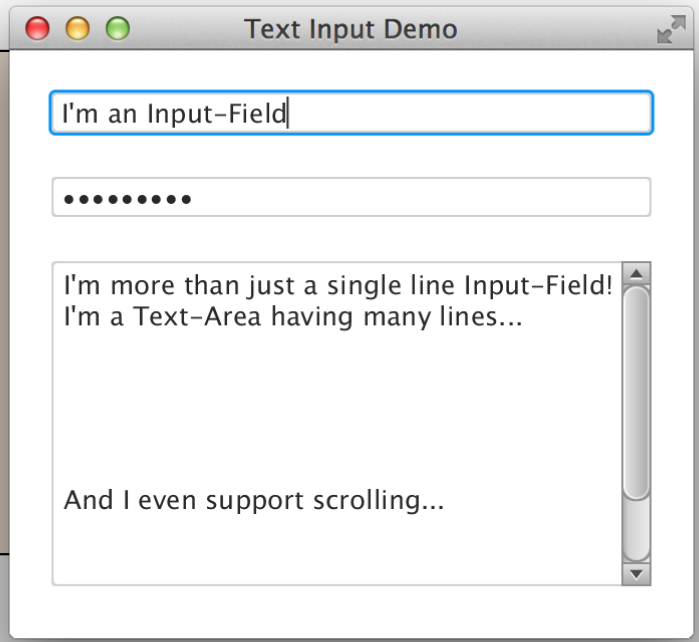
ButtonExample.java

Text Eingabefelder

- Es wird unterschieden zwischen einzeiligen Eingabefeldern (`TextField`) und mehrzeiligen (`TextArea`)
- Passwort-Felder (`PasswordField`) zeigen die Benutzereingabe nur verdeckt an, ansonsten sind es ganz normale Text-Felder

```
final TextField tf = new TextField();
root.getChildren().add(tf);

final PasswordField pf = new PasswordField();
root.getChildren().add(pf);

final TextArea ta = new TextArea();
ta.setWrapText(true);
root.getChildren().add(ta);
```

TextExample.java

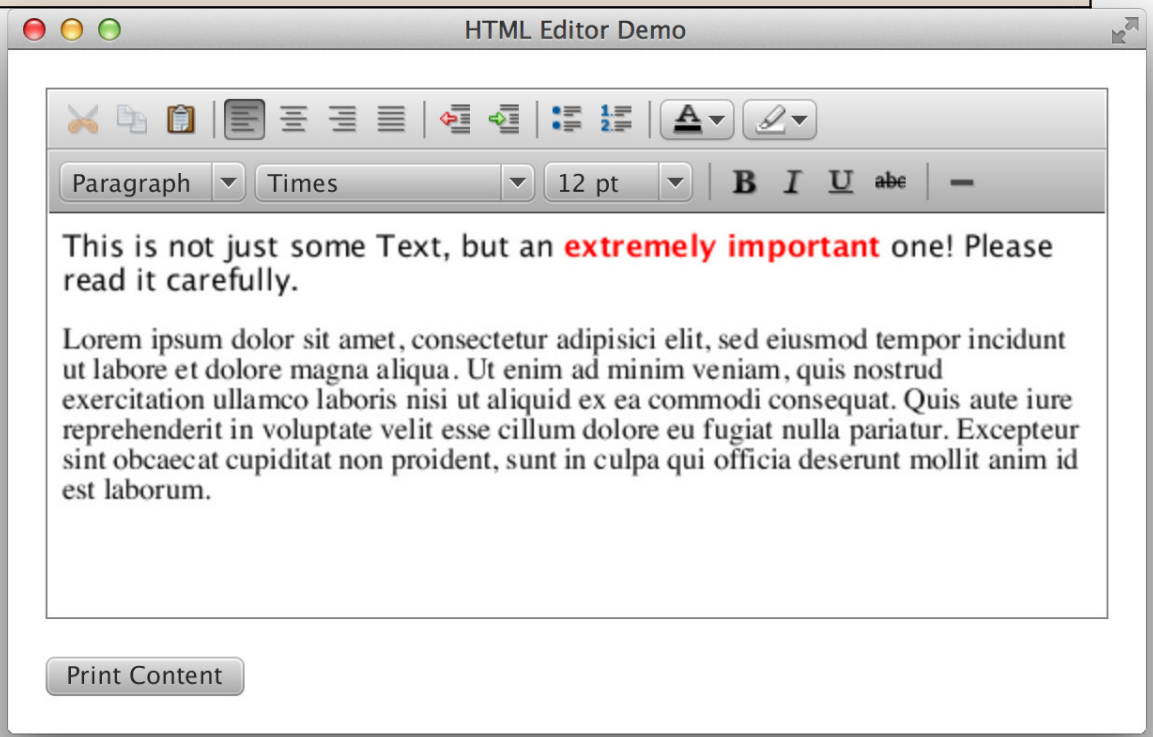


HTML Editor

- Für Texteingaben mit Formatierung
- Der ausgelesene Text ist in HTML

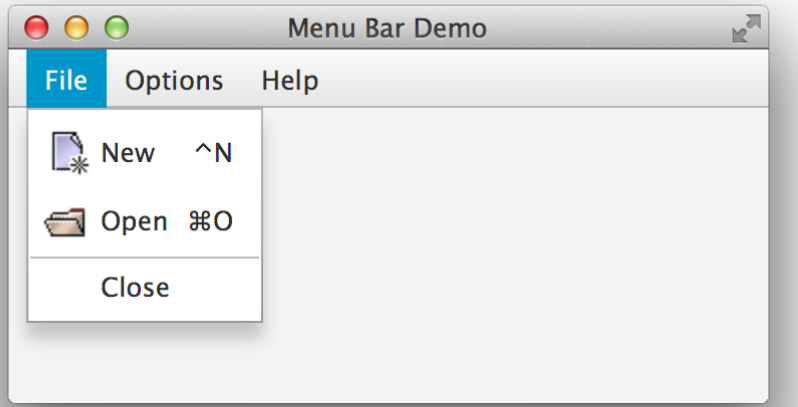
```
final HTMLEditor html = new HTMLEditor();
root.getChildren().add(html);
```

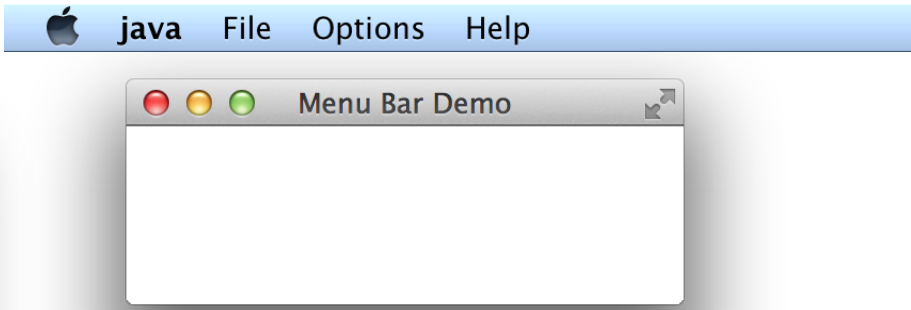
HTMLEditorExample.java



Menu-Bar

- Eine **Menu-Bar** kann in JavaFX überall platziert werden! Und muss explizit eingefügt werden
- **Einer** Menu-Bar kann das Property `useSystemMenuBar` gesetzt werden
- Eine Menu-Bar besteht aus **Menus** und diese wiederum aus **Menu-Items**





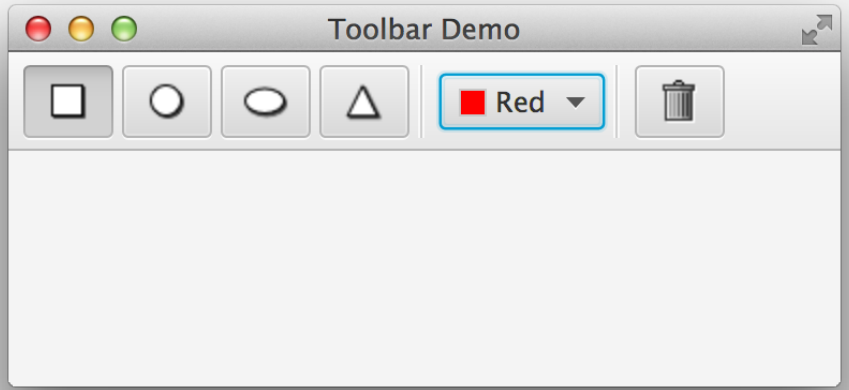
Menu-Bar

```
final MenuBar menuBar = new MenuBar();
menuBar.setUseSystemMenuBar(true);

final Menu fileM = new Menu("File");
final ImageView imgN = new ImageView(new Image("file:res/New.gif"));
final MenuItem newI = new MenuItem("_New", imageN);
newI.setAccelerator(KeyCombination.keyCombination("Ctrl+N"));
final ImageView imgO = new ImageView(new Image("file:res/Open.gif"));
final MenuItem openI = new MenuItem("_Open", imageO);
openI.setAccelerator(KeyCombination.keyCombination("Shortcut+O"));
final MenuItem closeI = new MenuItem("Close");

fileM.getItems().addAll(newI, openI, new SeparatorMenuItem(), closeI);
// ...
final Menu optionsM ...
final CheckMenuItem ...
final RadioMenuItem ...
// ...
menuBar.getMenus().addAll(fileM, optionM, helpM);

final VBox root = new VBox();
root.getChildren().addAll(menuBar);
```

MenuExample.java

Tool-Bar

- Eine **Tool-Bar** bietet einen schnellen Zugriff auf die am häufigsten verwendeten Befehle
- Enthält hauptsächlich Buttons, kann aber auch jegliche andere Nodes enthalten

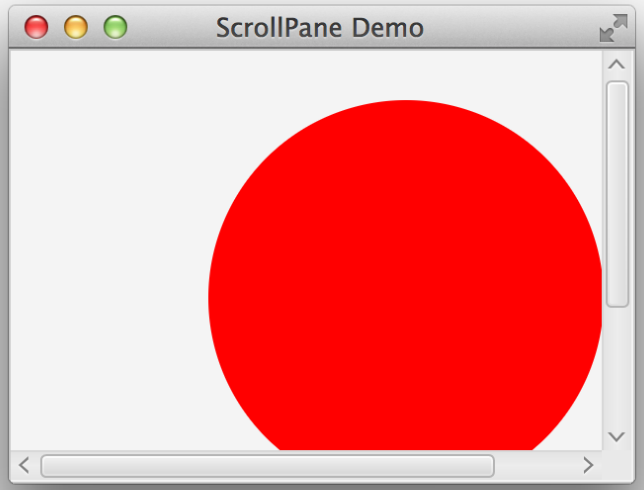


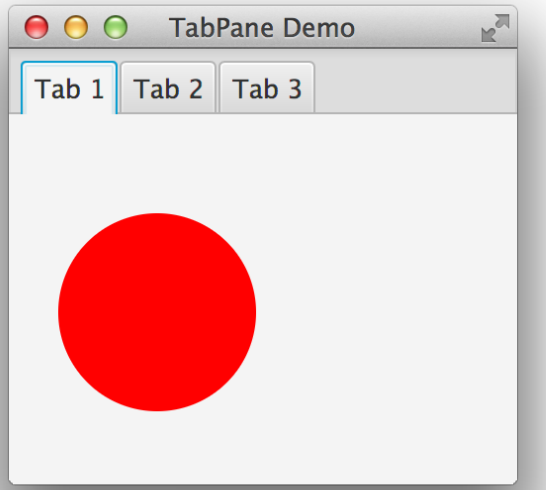
Tool-Bar

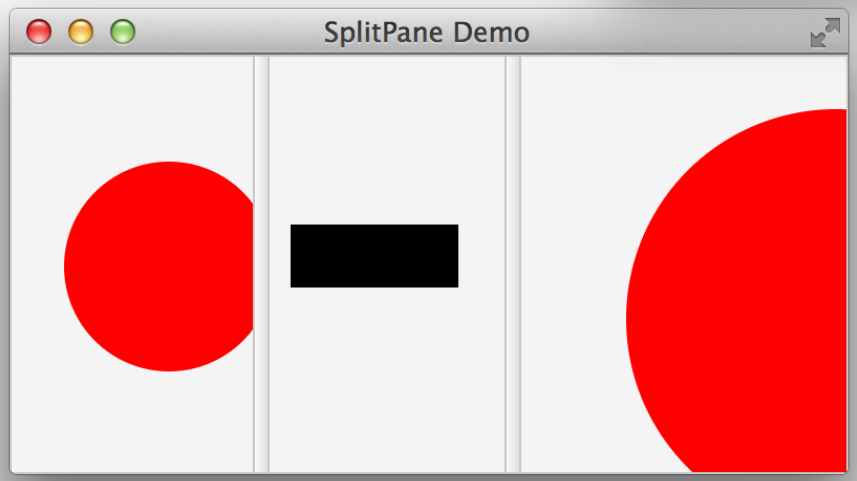
```
final ToolBar toolbar = new ToolBar();

final ToggleGroup toggleGroup = new ToggleGroup();
ToggleButton tb = new ToggleButton();
tb.setGraphic(new ImageView(new Image("file:res/Icon_Rectangle.png")));
tb.setToggleGroup(toggleGroup);
toolbar.getItems().add(tb);
// ...

final ColorPicker colorPicker = new ColorPicker(Color.RED);
toolbar.getItems().addAll(new Separator(), colorPicker);

final Button deleteButton = new Button(
       null, new ImageView(newImage("file:res/Delete.gif")));
toolbar.getItems().addAll(new Separator(), deleteButton);

final VBox root = new VBox();
root.getChildren().addAll(toolbar);
```

ToolBarExample.java

Scroll-/Split-/Tab-Pane







And so on...

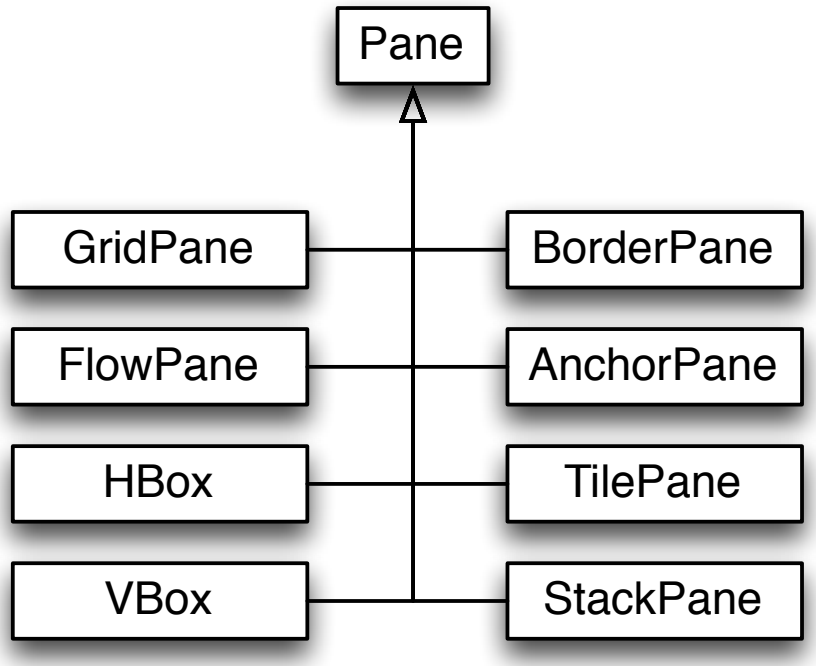
- ~~**https://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui_controls.htm#JFXUI336**~~

https://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui_controls.htm

Layout Panes

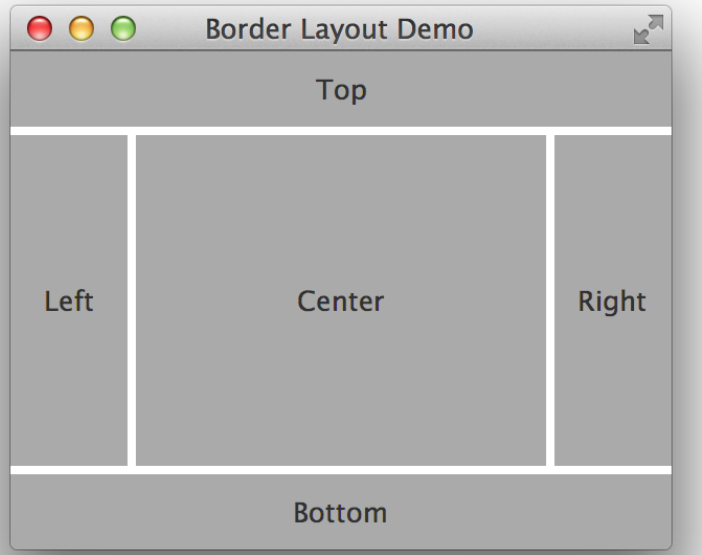
Layout Panes

- Um Controls innerhalb einer Scene auszurichten, bedient man sich Layout Panes
- Mit den zur Verfügung gestellten built-in Layout Panes können die meisten Konstrukte abgebildet werden



BorderPane

- Erzeugt einen grossen Zentralbereich (Center), der sich horizontal und vertikal ausdehnt
- Und vier Randbereiche(Top, Right, Bottom, Left), die sich nur horizontal, resp. vertikal ausdehnen
- Eignet sich oft als Grundlayout in einem Hauptfenster



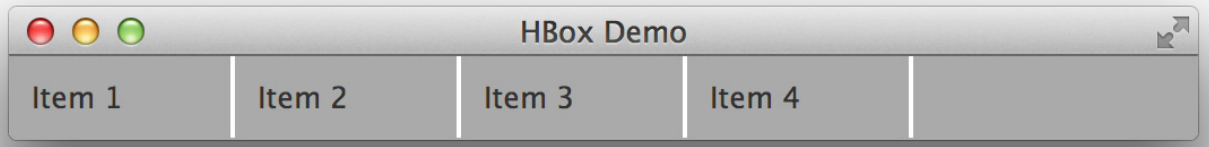
BorderPane

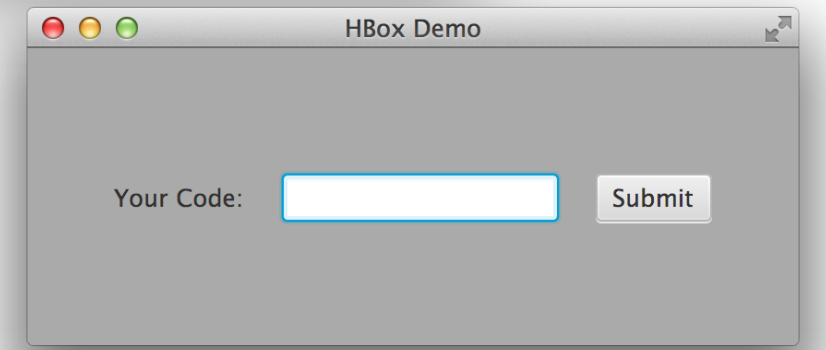
```
final BorderPane root = new BorderPane();

root.setTop(new Label("Top"));
root.setLeft(new Label("Left"));
root.setCenter(new Label("Center"));
root.setRight(new Label("Right"));
root.setBottom(new Label("Bottom"));
```

BorderPaneExample.java

HBox

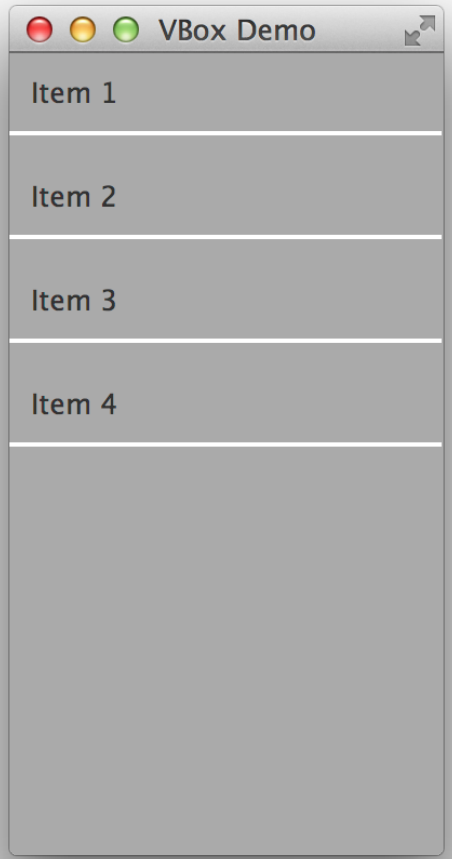
- Erzeugt einen horizontale Box und reiht alle Children nacheinander ein
- Nur eine Zeile, von links nach rechts oder von rechts nach links

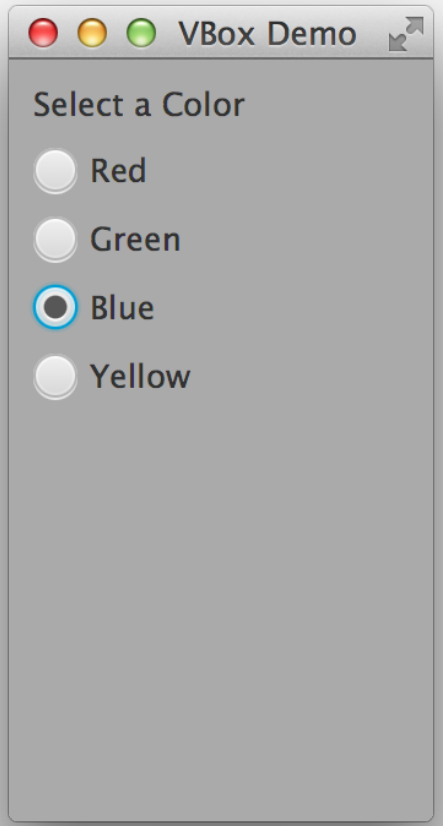




VBox

- Wie VBox, einfach in der Vertikalen





HBox/VBox

```
final HBox root = new HBox(20);

final Label label1 = new Label("Your Code:");
final TextField tf = new TextField();
final Button button = new Button("Submit");

root.getChildren().addAll(label1, tf, button);
```

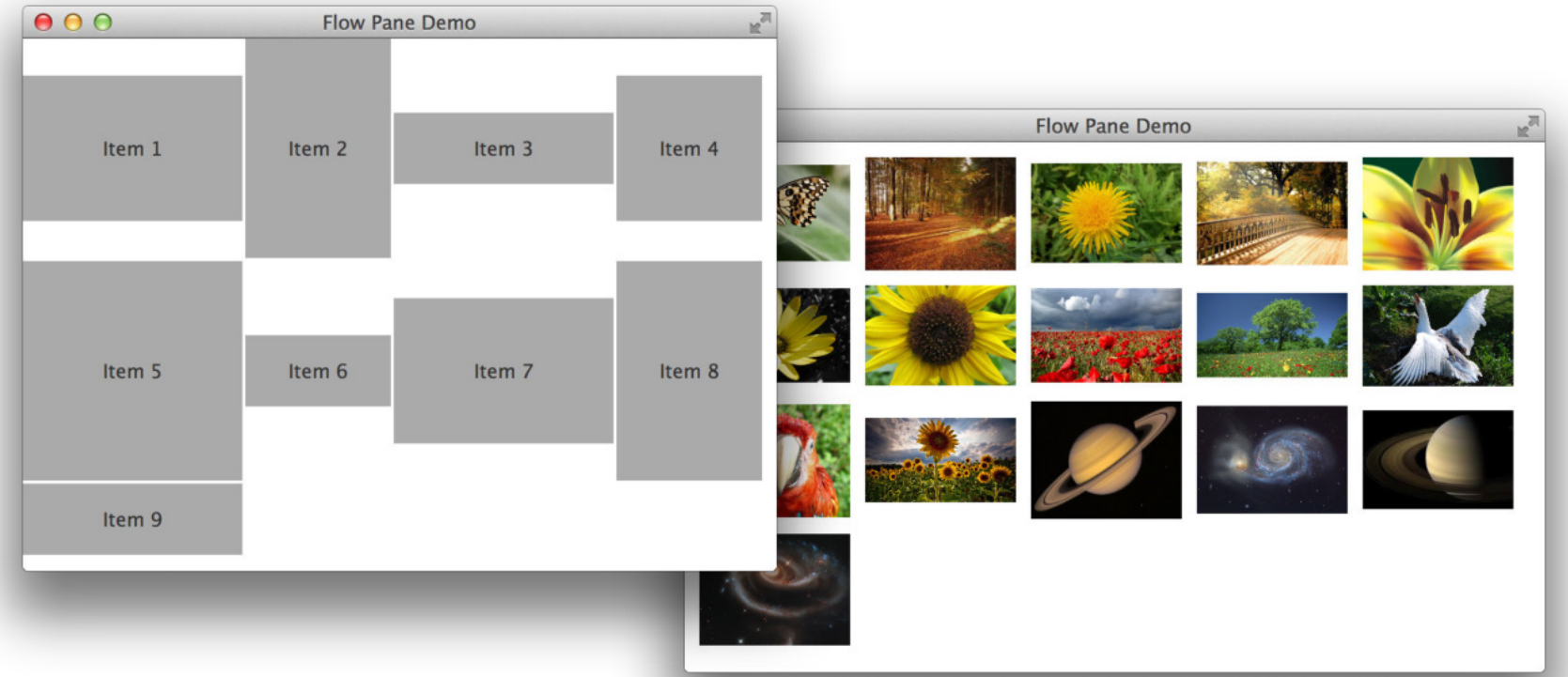
HBoxExample.java

```
final VBox root = new VBox(10);

final Label label1 = new Label("Select a Color");
final RadioButton rb1 = new RadioButton("Red");
final RadioButton rb2 = new RadioButton("Green");
final RadioButton rb3 = new RadioButton("Blue");
final RadioButton rb4 = new RadioButton("Yellow");

root.getChildren().addAll(label1, rb1, rb2, rb3, rb4);
```

VBoxExample.java

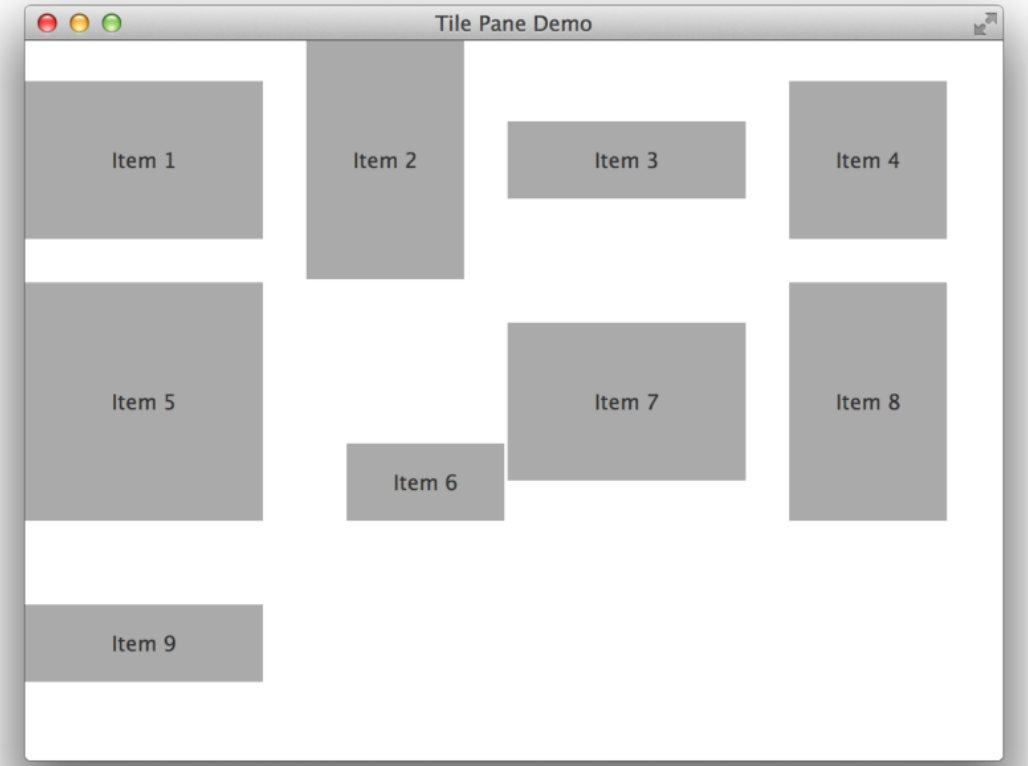
FlowPane

- Reiht alle Children nach einander ein
- Macht automatisch ’Zeilenumbrüche’
- Horizontal oder Vertikal, von links nach rechts, resp. von oben nach unten



TilePane

- Wie FlowPane, ausser dass jede Zelle (Tile) genau gleich gross ist
- Die Zellen sind so gross, damit der Inhalt der grössten Zelle Platz hat



FlowPane/TilePane

```
final FlowPane root = new FlowPane(10, 10);
for (int i = 0; i < 16; i++) {
    final ImageView image = new ImageView(
        new Image("file:res/image_" + i + ".jpg"));
    image.setPreserveRatio(true);
    image.setFitWidth(100);
    root.getChildren().add(image);
}
```

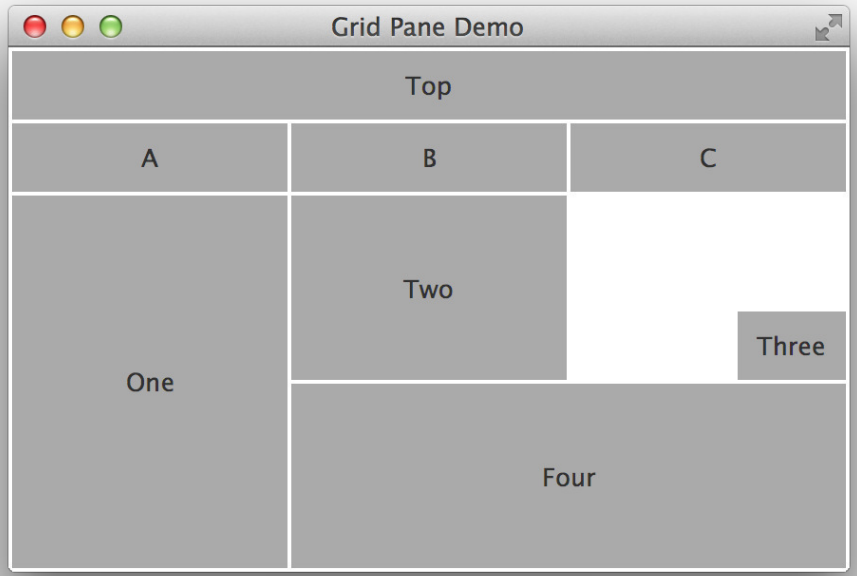
FlowPaneExample.java

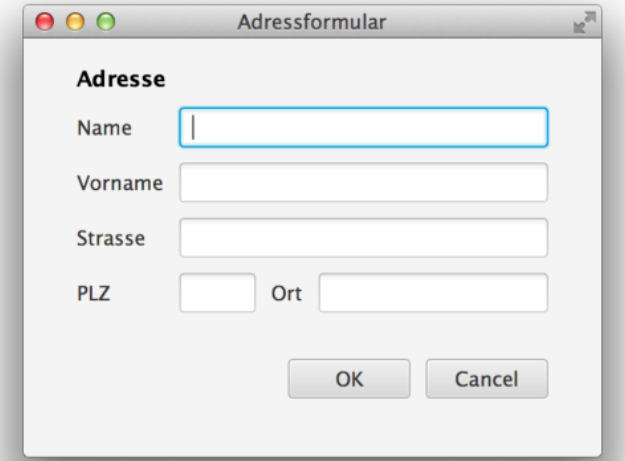
```
final TilePane root = new TilePane();
for (int i = 1; i <= 9; i++) {
    final Label label = new Label("Item " + i);
    //...
    if (i==6) {
        TilePane.setAlignment(label, Pos.BOTTOM_RIGHT);
    }
    root.getChildren().add(label);
}
```

TilePaneExample.java

GridPane

- Ist geeignet für komplexe Anordnungen
- Basiert auf Zeilen und Spalten
- Passt die Grösse von Zeilen und Spalten dynamisch an
- Zellen können leer bleiben
- Eine Zelle kann mehrere Spalten oder Zeilen überspannen
- Eine Zelle muss nicht ganz ausgefüllt werden





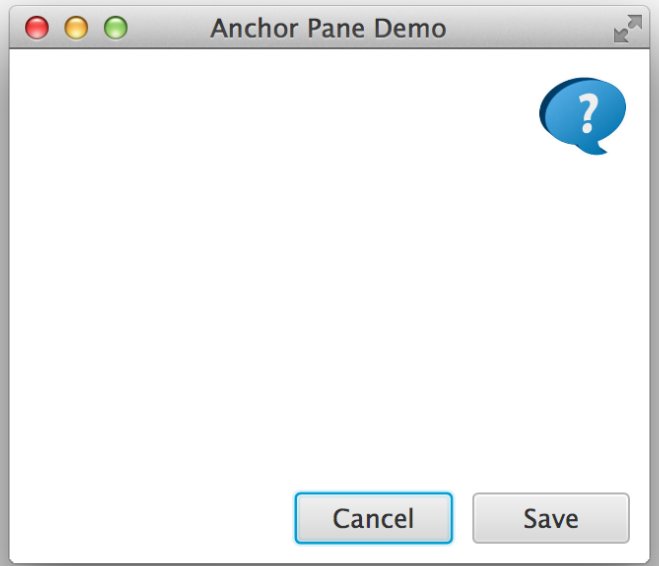
GridPane

```
final GridPane root = new GridPane();

final Label label1 = new Label("Top");
GridPane.setConstraints(label1, 0, 0, GridPane.REMAINING, 1);

final Label label2 = new Label("A");
GridPane.setConstraints(label2, 0, 1);

// ...

final Label label7 = new Label("Three");
GridPane.setConstraints(label7, 1, 2, 2, 1, HPos.RIGHT, VPos.BOTTOM);

root.getChildren().addAll(label1, label2, ...);
```

GridPaneExample.java

AnchorPane

- Sehr praktisch, um Knöpfe und Icons zu verankern!



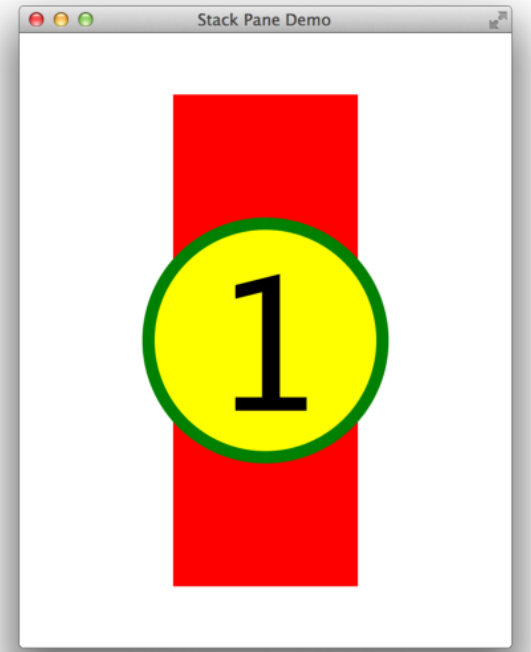
AnchorPane

```
final AnchorPane root = new AnchorPane();

Button saveB = new Button("Save");
AnchorPane.setBottomAnchor(saveB, 10.0);
AnchorPane.setRightAnchor(saveB, 10.0);

Button cancelB = new Button("Cancel");
AnchorPane.setBottomAnchor(cancelB, 10.0);
AnchorPane.setRightAnchor(cancelB, 100.0);

final ImageView help = new ImageView(new Image("file:res/help.jpg"));
help.setPreserveRatio(true);
help.setFitHeight(48);
AnchorPane.setTopAnchor(help, 10.0);
AnchorPane.setRightAnchor(help, 10.0);

root.getChildren().addAll(cancelB, saveB, help );
```

AnchorPaneExample.java

StackPane

- Alle Children werden übereinander gelegt; in der Reihenfolge wie sie eingefügt wurden



StackPane

```
final StackPane root = new StackPane();

final Rectangle rect1 = new Rectangle(150, 400, Color.RED);
final Circle circ1 = new Circle(100, Color.GREEN);
final Circle circ2 = new Circle(90, Color.YELLOW);
final Text text1 = new Text("1");
text1.setFont(Font.font(150));
final Rectangle rect2 = new Rectangle(200, 450, Color.GREY);

root.getChildren().addAll(rect1, circ1, circ2, text1, rect2);
```

StackPaneExample.java

Berechnung der Grössen

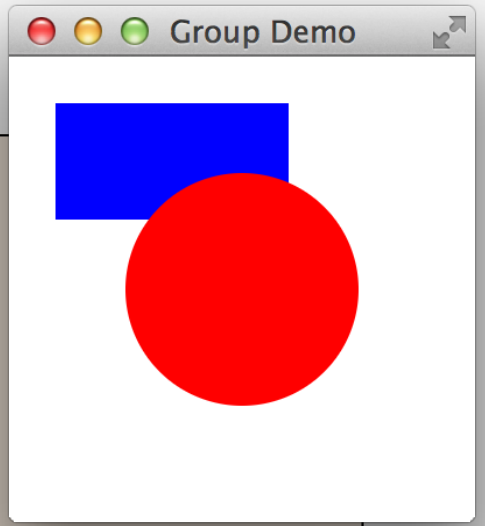
Die komplexe Berechnung der Grössen eines Layout Panes und deren Children basiert auf folgenden Werten

- `minWidth` / `minHeight`
- `prefWidth` / `prefHeight`
- `maxWidth` / `maxHeight`

Auf das explizite Setzen des `width`/`height` Property sollte verzichtet werden

Group

- Für das lose Zusammenfügen von Nodes
- Die Group ist nicht resizable, sondern passt ihre Grösse dem Inhalt an
- Kann mit dem Gruppieren eines Zeichenprogramms verglichen werden




```
final Group root = new Group();

final Rectangle rect = new Rectangle(20, 20, 100, 50);
rect.setFill(Color.BLUE);
final Circle circ = new Circle(100, 100, 50);
circ.setFill(Color.RED);

root.getChildren().addAll(rect, circ);
```

GroupExample.java

Application

Application

```
public class MyApp extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        Parent root = ...

        // Create GUI

        Scene scene = new Scene(root);
        stage.setTitle("My App");
        stage.setScene(scene);
        stage.show();
    }

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

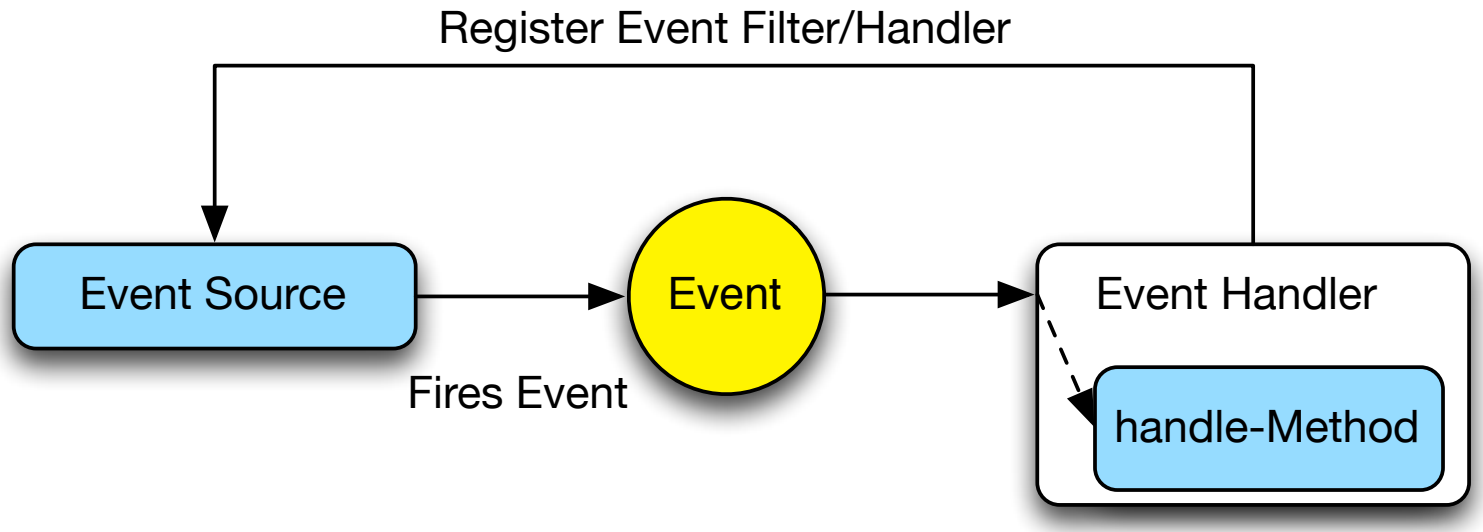
MyApp.java

Events

Events

- Die Interaktion mit einem GUI geschieht über Events
- Events werden durch Benutzeraktionen (Drücken eines Buttons, Verschieben der Maus, Drücken einer Taste, ...) ausgelöst
- Um programmatisch auf Aktionen eines Benutzers zu reagieren, können Event-Filter und -Handler registriert werden

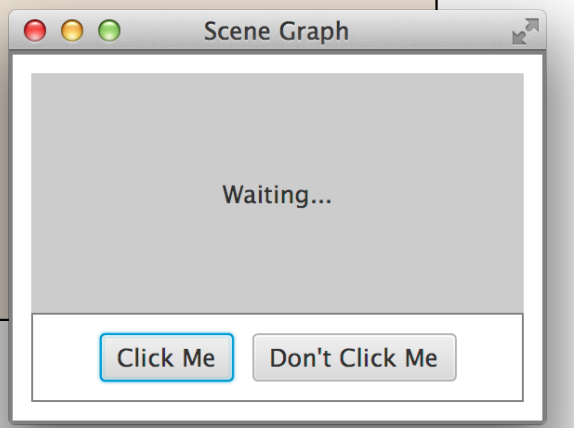
Events


Register Event Filter/Handler
Event Source
Event
Fires Event
Event Handler
handle-Method

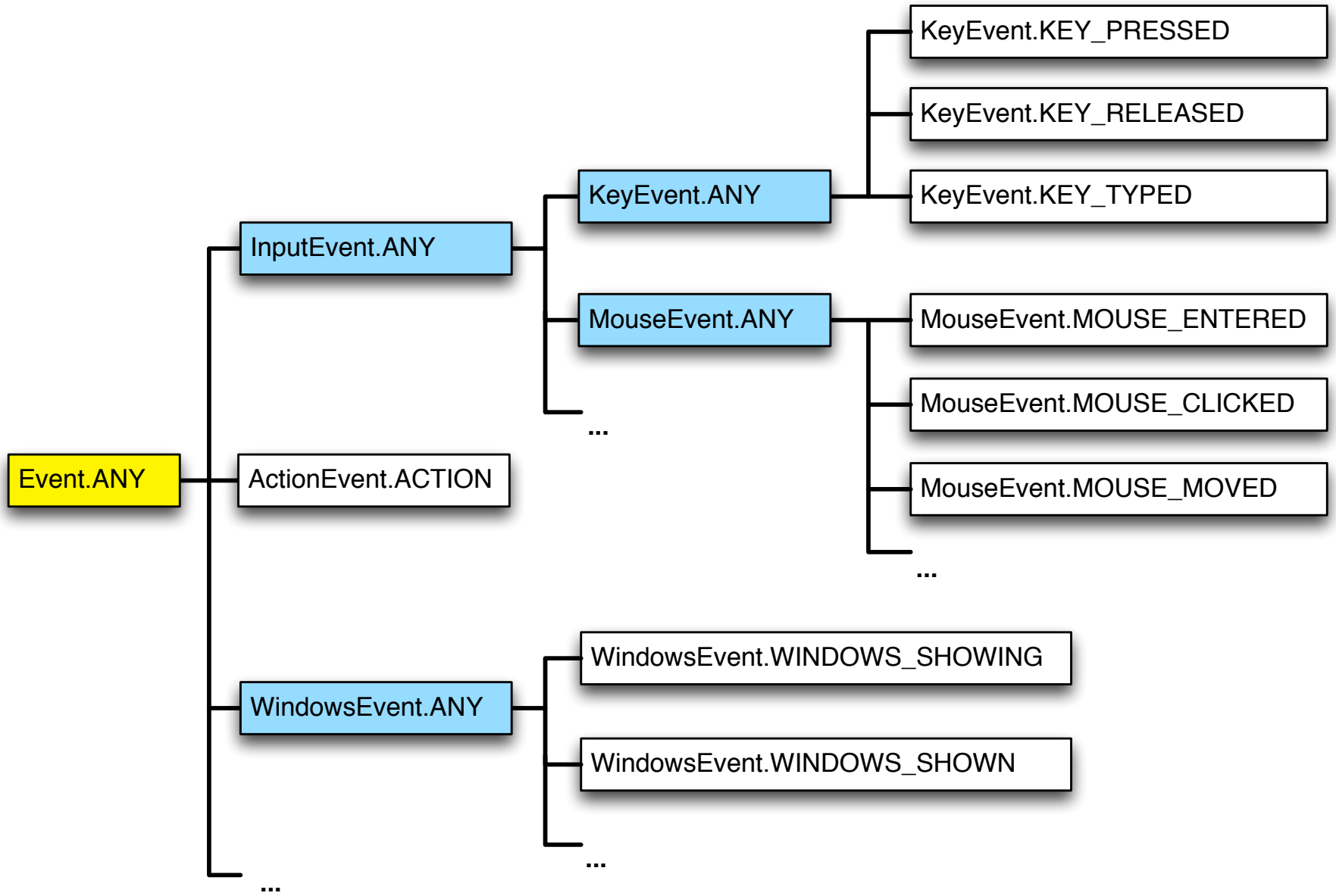
Example

```
public class SceneDemo extends Application
                       implements EventHandler<ActionEvent> {

    private Label label;
    ...

    @Override
    public void start(Stage stage) throws Exception {
        ...
        Button button = new Button("Click Me");
        button.addEventHandler(ActionEvent.ACTION, this);
        ...
    }

    @Override
    public void handle(ActionEvent event) {
        this.label.setText("Hello World!");
    }
}
```

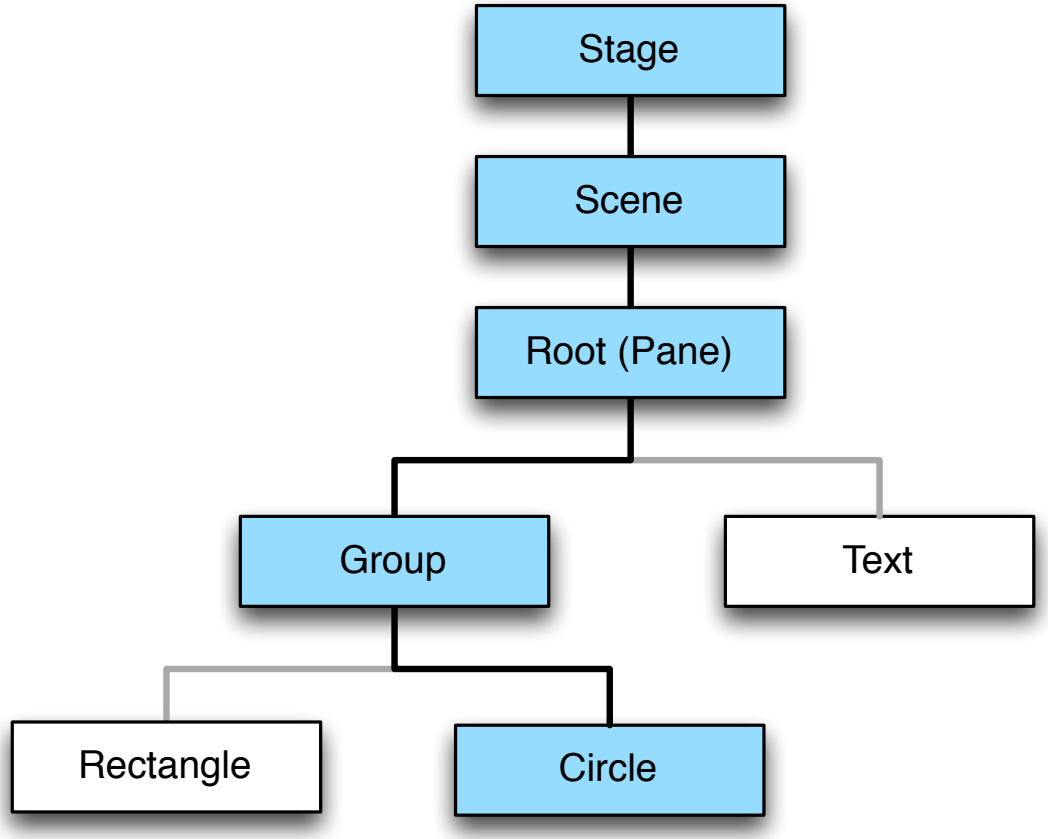
SceneDemo.java

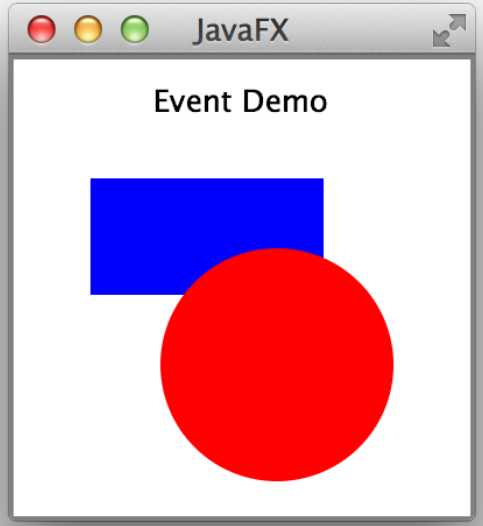


Event Types


Event.ANY
InputEvent.ANY
KeyEvent.ANY
KeyEvent.KEY_PRESSED
KeyEvent.KEY_RELEASED
KeyEvent.KEY_TYPED
...
MouseEvent.ANY
MouseEvent.MOUSE_ENTERED
MouseEvent.MOUSE_CLICKED
MouseEvent.MOUSE_MOVED
...
ActionEvent.ACTION
WindowsEvent.ANY
WindowsEvent.WINDOWS_SHOWING
WindowsEvent.WINDOWS_SHOWN
...
...

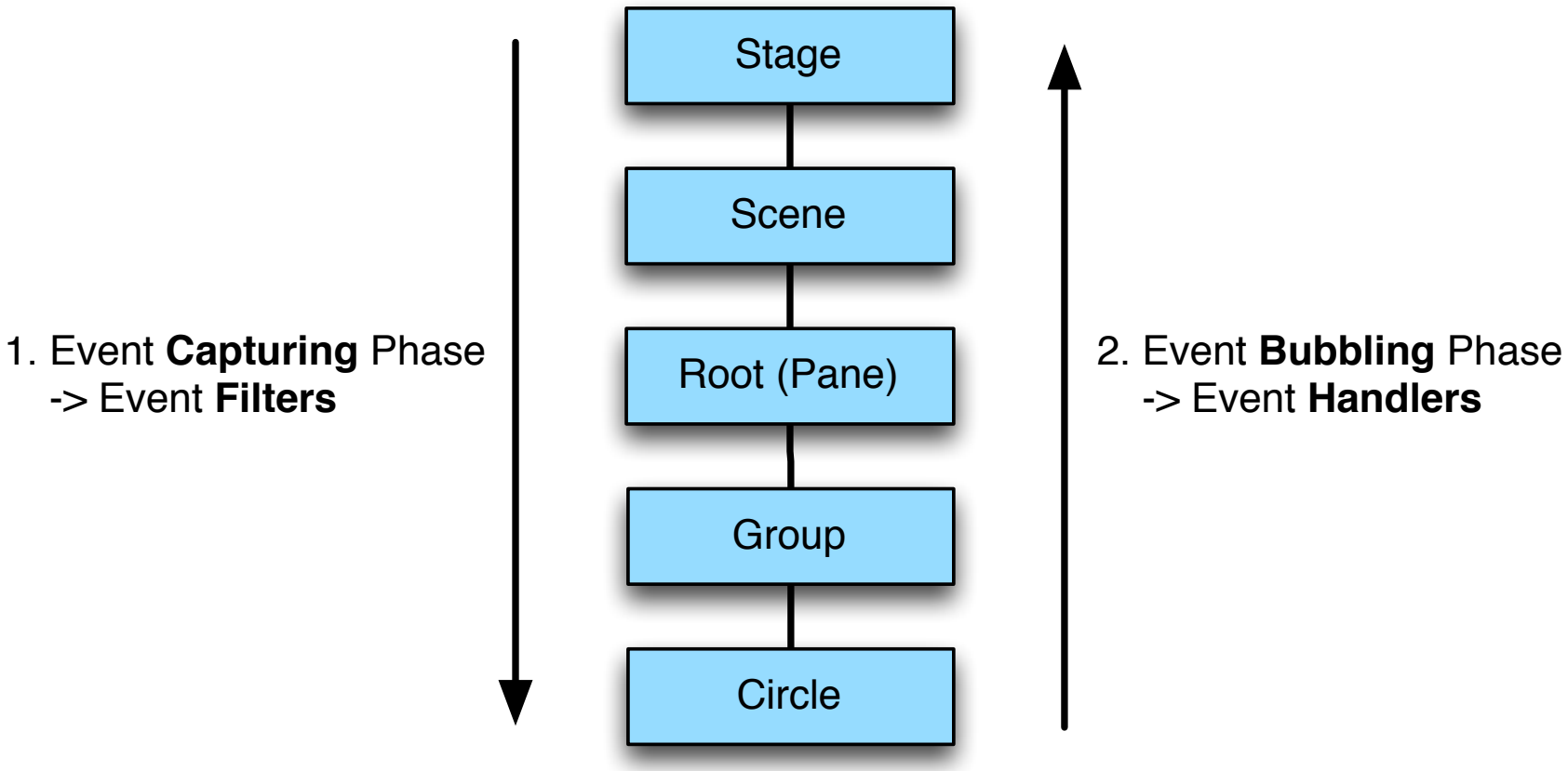
Event Dispatch Chain





Initiale Route-Konstruktion von der Stage zum Target-Node, bei einem Click auf den roten Kreis

Event Propagation



Jeder registrierte Event-Filter und -Handler kann den Event konsumieren (consume) und verhindert dadurch das weitere propagieren des Events

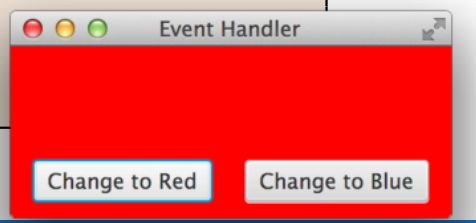
Event-Filters/-Handlers

- Event-Handler sowie auch Event-Filter sind Implementationen des EventHandler Interface
- Das EventHandler Interface hat eine Methode: handle()
- Ein Event-Filter wird mittels addEventFilter() registriert und mit removeEventFilter() wieder entfernt
- Dito für Event-Handler
- Event-Filter/-Handler können auf vier verschiedene Arten implementiert werden:
  - Durch eine Klasse (zB. der Main-Klasse)
  - Durch eine innere Klasse
  - Durch eine anonyme Klasse
  - Durch einen Lambda-Ausdruck

Event-Filters/-Handlers

```
public class EventDemo extends Application
                        implements EventHandler<ActionEvent>{

    private BorderPane root;
    private Button red, blue;

    @Override public void start(Stage stage) throws Exception {
        ...
        this.red = new Button("Change to Red");
        this.red.addEventHandler(ActionEvent.ACTION, this);

        this.blue = new Button("Change to Blue");
        this.blue.addEventHandler(ActionEvent.ACTION, this);
    }
    @Override public void handle(ActionEvent event) {
        if ( event.getSource() == this.red ) {
            this.root.setStyle("-fx-background-color: red");
        } else if ( event.getSource() == this.blue ) {
            this.root.setStyle("-fx-background-color: blue");
        }
    }
}
```

EventDemo.1.java



Event-Filters/-Handlers

```
public class EventDemo extends Application {

    private BorderPane root;

    @Override public void start(Stage stage) throws Exception {
        ...
        final Button red = new Button("Change to Red");
        red.addEventHandler(ActionEvent.ACTION, new ChangeBg("red"));

        final Button blue = new Button("Change to Blue");
        blue.addEventHandler(ActionEvent.ACTION, new ChangeBg("blue"));
    }

    // Inner class
    private class ChangeBg implements EventHandler<ActionEvent>{
        private final String color;
        public ChangeBg(String color) { this.color = color; }

        @Override public void handle(ActionEvent event) {
            root.setStyle("-fx-background-color: " + this.color);
        }
    }
}
```

EventDemo.2.java

Event-Filters/-Handlers

```
public class EventDemo extends Application {

    @Override public void start(Stage stage) throws Exception {

        ...

        final Button red = new Button("Change to Red");
        // Anonymous class
        red.addEventHandler(ActionEvent.ACTION,
                new EventHandler<ActionEvent>() {
                    @Override
                    public void handle(ActionEvent event) {
                        root.setStyle("-fx-background-color: red");
                    }
                });

        final Button blue = new Button("Change to Blue");
        // Lambda expression
        blue.addEventHandler(ActionEvent.ACTION,
                event -> root.setStyle("-fx-background-color: blue"));
    }
}
```

EventDemo.3.java

Event-Filters/-Handlers

- Mehrere Event-Handler/-Filter können für einen Event registriert weden
- Ein Event-Handler/-Filter kann für mehrere Events registriert werden
- Sind mehrere Handler/Filter auf auf einem Node registriert, werden sie aufgrund der Event-Type-Hierarchie aufgerufen (je spezifischer, je früher)
- Es gibt eine Vielzahl von convenience Methoden, die es erlauben, für einen bestimmten Event genau einen Event-Handler zu registrieren
  - `setOnAction()`
  - `setOnMouseClicked()`
  - `setOnKeyPressed()`
  - `...`

Event Objekt

- Das Event-Objekt liefert Informationen über das erfolgte Ereignis
- getTarget() gibt das Objekt zurück, auf dem das Ereignis ausgelöst wurde
- getSource() gibt das Objekt zurück, auf dem der Event-Handler/-Filter registriert wurde
- consume() konsumiert den Event (i.e. stoppt die weitere Propagation)
- Viele Event spezifische Informationen wie Mausposition, Keycode, etc.