

# JavaFX

- Framework/Classlibrary für GUI
- Entwicklungsgeschichte ist sehr inhomogen und noch nicht abgeschlossen.
- Ermöglicht Gestensteuerung (Drehen, zoomen, ...)
- Arbeitet, wenn möglich, direkt mit der Grafikkarte zusammen (DirectX, OpenGL)
- GUI kann in Java oder FXML programmiert werden
- Letzteres ermöglicht gestalterische Änderungen ohne neu zu kompilieren
- Beschaffung:
  - <http://www.oracle.com/technetwork/java/javafxscenebuilder-1x-archive-2199384.html>

# JavaFX

## JavaFX unter Java11

- JavaFx ist nicht Bestandteil des jdk11
- Download von:
  - <https://gluonhq.com/products/javafx/>
- Compilieren mit:
  - `javac --module-path /opt/javafx-sdk-11.0.2/lib/ --add-modules=javafx.controls,javafx.fxml Hello1.java`  
<https://gluonhq.com/products/javafx/>
- Ausführen mit:
  - `java --module-path /opt/javafx-sdk-11.0.2/lib/ --add-modules=javafx.controls,javafx.fxml Hello1`
- Oder Java8 benutzen, da ist Fx inclusive

# JavaFX

Ein Tutorial zum Einstieg:

[http://docs.oracle.com/javafx/2/get\\_started/jfxpub-get\\_started.htm](http://docs.oracle.com/javafx/2/get_started/jfxpub-get_started.htm)



# JavaFX

## Programmaufbau:

- Hauptklasse muss von `javafx.application.Application` abgeleitet sein, man spricht auch von der Starterklasse
- Es kann eine `main`-Funktion geben, muss es aber nicht.

- Eine Methode

```
@Override public void start(Stage primaryStage)
```

ist notwendig, sie wird automatisch aufgerufen.

- `PrimaryStage` ist die Hauptbühne, auf der sich alles abspielt.
- Alle Komponenten der Oberfläche bilden die Szene (`scene`)
- Intern werden die Komponenten als `Nodes` einer Liste verwaltet, wobei jeder `Node` wieder eine Liste enthalten kann. So entsteht der `sceneTree`.

# JavaFX

Ein minimales Hello-Programm:

```
public class Hello1 extends Application
{
    @Override
    public void start(Stage primaryStage)
    {
        Button btn = new Button("Say 'Hello World'");
        Scene scene = new Scene(btn, 300, 250);

        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

Die Szene besteht hier nur aus dem Button, Üblicherweise eher aus einem Container, in JavaFX Pane genannt.



# JavaFX

Dazu die notwendigen import-Anweisungen:

```
import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.stage.Stage;
```

So sollte es auch gehen:

```
import javafx.application.*;  
import javafx.scene.*;  
import javafx.scene.control.*;  
import javafx.stage.*;
```

# JavaFX

Wir ergänzen einen Eventhandler (klassisch mit ananymer Klasse):

```
import javafx.event.*;
. . .
Button btn = new Button();
btn.setText("Say 'Hello World'");

btn.setOnAction(new EventHandler<ActionEvent>()
{
    @Override
    public void handle(ActionEvent event)
    {
        System.out.println("Hello World!");
    }
});
```

Attribute werden oft via  
Setter eingestellt.

EventHandler haben nur  
eine Funktion  
(funktionales Interface)

Oder Eventhandler in Form eines **Lambda Ausdrucks**:

```
btn.setOnAction(event->System.out.println("Hello World!"));
```

# JavaFX

Soll in einem Lambda Ausdruck\* eine Funktion gerufen werden,

```
void myAction(ActionEvent e)
{
    System.out.println("Hello World!");
}
. . .
btn.setOnAction(e->myAction(e));
```

So kann die Methode auch in Form einer „**Method Reference**“ als Eventhandler übergeben werden.

```
void myAction(ActionEvent e){System.out.println("Hello World!");}
. . .
btn.setOnAction(this::myAction);
```

\*) Lambda Ausdruck scheint etwas unpassend formuliert zu sein, es folgt eine Anweisung, sogar ein Block kann folgen.



# JavaFX

Wir ergänzen einen Container:

```
import javafx.scene.layout.*;
. . .
StackPane root = new StackPane();
root.getChildren().add(btn);

Scene scene = new Scene(root, 300, 250);
```

`getChildren()` liefert eine Liste der vom Container verwalteten Nodes (Node entspricht in etwa einer Komponente in AWT). An diese Liste wird ein oder werden mehrere Element(e) angehängen. Die Liste kann leer, aber nicht null sein.

`StackPane` ordnet die Nodes wie ein `CardLayout` hintereinander an. Sichtbar ist das zuletzt in die Liste eingefügte Node. Über die Listenoperationen `remove` und `add` wird gesteuert, welches Node zu sehen ist. Ist ein Node transparent, sieht man auch das darunterliegende Node.

→ Beispiel `FXStackpPane`

# JavaFX

So könnte die start-Methode nun aussehen:

```
@Override
public void start(Stage primaryStage)
{
    Button btn = new Button();
    btn.setText("Say 'Hello World'");
    StackPane root = new StackPane();
    Scene scene = new Scene(root, 300, 250);

    btn.setOnAction(e->{System.out.println("Hello World!");});

    root.getChildren().add(btn);

    primaryStage.setTitle("Hello World!");
    primaryStage.setScene(scene);
    primaryStage.show();
}
```



# JavaFX

Um auch hier wiederverwendbare Komponenten zu erhalten, teilen wir den Quelltext wie folgt:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.*;

public class Hello3Main extends Application
{
    @Override
    public void start(Stage primaryStage)
    {
        Hello3 root = new Hello3();
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

Hier haben wir  
den Code ausgelagert

# JavaFX

Hier jetzt der wiederverwendbare Teil:

```
import javafx.scene.control.*;
import javafx.event.*;
import javafx.scene.layout.*;

public class Hello3 extends StackPane
{
    public Hello3()
    {
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.setOnAction(e->{System.out.println("Hello World!");});
        getChildren().add(btn);
    }
}
```

# JavaFX

Commandline Argumente:

Von der main Funktion werden die Kommandozeilenargumente über `launch(args)` weitergeleitet. Mit Hilfe der Methode `getParameters` des Applicationobjektes bekommt man ein Objekt der Klasse `Application.Parameters`, in welchem die Parameter als List und als Map verwaltet werden. Mit Hilfe der Methoden von List kann man auf die Parameter dann zugreifen.

Im Beispiel wird der Button mit dem ersten Text des ersten Kommandozeilenparameters versehen. Dies geschieht hier noch in der `start`-Methode.

```
String s=getParameters().getRaw().get(0);  
btn.setText(s);
```

# JavaFX

Die Container:

```
import javafx.scene.layout.*;
```

- BorderPane
- HBox
- VBox
- StackPane
- GridPane
- FlowPane
- TilePane
- AnchorPane
- Pane

# JavaFX

Als Experimentiergrundlage benutzen wir nachfolgende Applikationsklasse. Über Kommandozeilenparameter können wir steuern, welches Pane wir testen wollen.

```
public class TestContainer extends Application
{
    public static void main(String[] args)
    { launch(args); }

    @Override public void start(Stage primaryStage)
    {
        String s="";
        Pane p=null;
        switch(s=getParameters().getRaw().get(0))
        {
            case "Border": p=new FXBorderPane(); break;
        }
        primaryStage.setTitle("JavaFX Welcome");
        Scene scene = new Scene(p);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

Hier für weitere  
Beispiele  
Case-Zweige  
Ergänzen  
(bemerke: Strings  
Als CaseLabel!)

```
import javafx.application.*;
import javafx.scene.Scene;
import javafx.scene.layout.*;
import javafx.stage.*;
```

# JavaFX

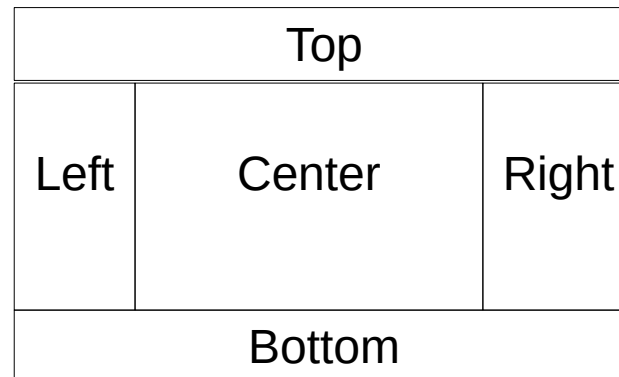
BorderPane:

Ähnlich einem Panel mit BorderLayout

Es können max. 5 Nodes eingetragen werden

Zum Einsetzen in die 5 Zonen gibt es die Funktionen:

- `setTop`
- `setBottom`
- `setLeft`
- `setRight`
- `setCenter`





# JavaFX

```
public class FXBorderPane extends BorderPane
{
    Button b1=new Button("Prof. Beck, Z356, Tel. 2130");
    Button b2=new Button("rechts vom Beck");
    Button b3=new Button("beck@informatik.htw-dresden.de");
    Button b4=new Button("links vom Beck");
    ImageView iv=new ImageView(new Image("beck.JPG"));
```

```
public FXBorderPane()
{
    b1.setMaxWidth(99999);
    setTop(b1);
    b3.setMaxWidth(Double.MAX_VALUE);
    setBottom(b3);
    b2.setMaxHeight(999.9);
    setRight(b2);
    b4.setMaxHeight(999.9);
    setLeft(b4);
    setCenter(iv);
}
}
```

Großer Wert! Bewirkt das Aufweiten des Node auf die Größe der Zelle



# JavaFX

## Images:

In JavaFX werden Bilder, eingebettet in ein Objekt der Klasse ImageView wie gewöhnliche Nodes dargestellt.

<https://docs.oracle.com/javafx/2/api/javafx/scene/image/ImageView.html>

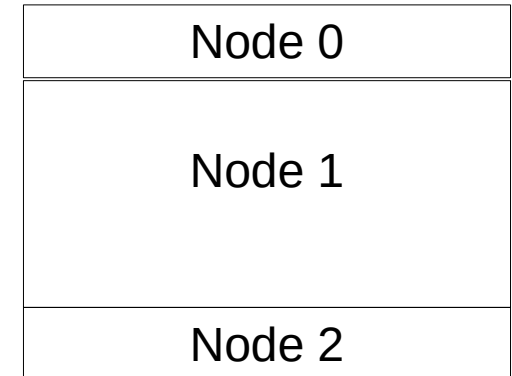
- Es können Ausschnitte gebildet werden.
- Das Bild kann rotiert werden.
- Das Bild kann scaliert (auch vergrößert) werden.

```
iv.setFitWidth(300);  
iv.setPreserveRatio(true);  
iv.setSmooth(true);  
iv.setCache(true);
```

# JavaFX

## Vbox/HBox:

- Vertikales/Horizontales Boxlayout
- Defaultbreite/-höhe wird von breitem/höchstem Node bestimmt.
- Verarbeitung mittels Listfunktionen
  - `public ObservableList<Node> getChildren()`
  - Liefert eine Liste der Nodes, die die VBox verwaltet.
  - Diese Liste kann leer sein, aber nicht null.
  - Über `remove` und `add` können Nodes entfernt/hinzugefügt werden.



```
getChildren().add(bs[i]);  
getChildren().addAll(bs[0], bs[1], bs[2], bs[3], bs[4]);
```

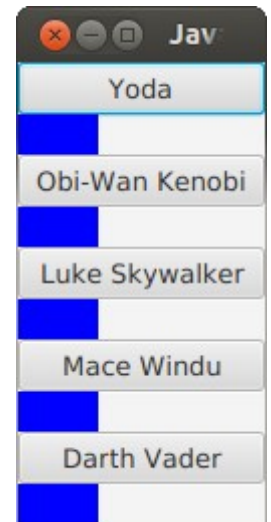


# JavaFX

```
public class FXVBox extends VBox
{
    String lbls[]= {"Yoda","Obi-Wan Kenobi",
                  "Luke Skywalker","Mace Windu","Darth Vader"};
    Button bs;

    public FXVBox()
    {
        for(int i=0; i<lbls.length;i++)
        {
            bs=new Button(lbls[i]); bs.setMaxWidth(9999);
            bs.setOnAction(e->
                System.out.println(((Button)e.getSource()).getText()));
            getChildren().add(bs);
            getChildren().add(new Rectangle(40,20,Color.BLUE));
        }
        //getChildren().addAll(bs[0],bs[1],bs[2],bs[3],bs[4]);
    }
}
```

```
import javafx.scene.shape.*;
import javafx.scene.paint.*;
```



Auch graphische  
Elemente können  
mit add  
eingefügt  
werden

# JavaFX

```
public class FXHBox extends HBox
```

```
{
```

```
    String lbls[]={ "Yoda", "Obi-Wan Kenobi", "Luke Skywalker",  
                    "Mace Windu", "Darth Vader" };  
    Button bs;
```

```
    public FXHBox()
```

```
{
```

```
    for(int i=0; i<lbls.length;i++)
```

```
{
```

```
        bs=new Button(lbls[i]); bs.setMaxHeight(9999);
```

```
        bs.setOnAction(e->
```

```
            System.out.println(((Button)e.getSource()).getText()));
```

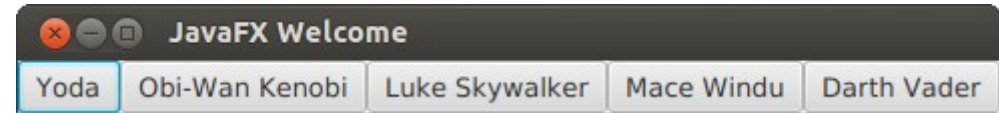
```
        getChildren().add(bs);
```

```
}
```

```
    //getChildren().addAll(bs[0],bs[1],bs[2],bs[3],bs[4]);
```

```
}
```

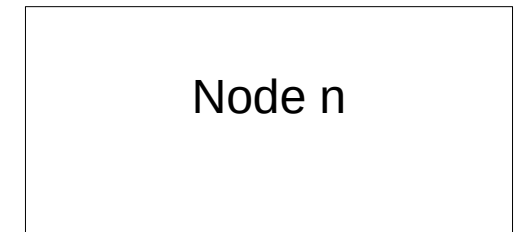
```
}
```



# JavaFX

## FXStackPane:

- Entspricht in etwa einem Panel mit CardLayout.
- Es ist immer nur das letzte Node der NodeList zu sehen.
- Defaultbreite/-höhe wird von breitem/höchstem Node bestimmt.
- Verarbeitung mittels Listfunktionen
  - `public ObservableList<Node> getChildren()`
  - Liefert eine Liste der Nodes, die die VBox verwaltet.
  - Diese Liste kann leer sein, aber nicht null.
  - Über `remove` und `add` können Nodes entfernt/hinzugefügt werden.

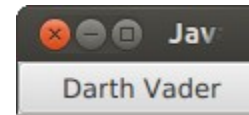


```
getChildren().add(bs[i]);  
getChildren().addAll(bs[0], bs[1], bs[2], bs[3], bs[4]);
```

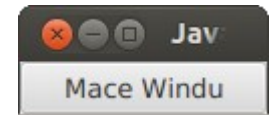
# JavaFX

```
public class FXStackPane extends StackPane
{
    String lbls[]=
    {"Yoda", "Obi-Wan Kenobi", "Luke Skywalker", "Mace Windu", "Darth Vader"};
    Button bs;
    void myHandleAction(ActionEvent e)
    {
        Button btn=(Button)e.getSource();
        System.out.println(btn.getText());
        this.getChildren().remove(btn);
        this.getChildren().add(0, btn);
    }
    public FXStackPane()
    {
        for(int i=0; i<lbls.length; i++)
        {
            bs=new Button(lbls[i]); bs.setMaxWidth(9999);
            bs.setOnAction(e->myHandleAction(e));
            getChildren().add(bs);
        }
    }
}
```

Letztes Element aus der Nodeliste entfernen und vorn wieder einfügen



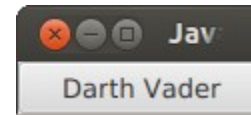
Click->



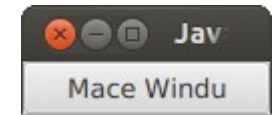
Eventhandling mit Lambda-Ausdruck

# JavaFX

```
public class FXStackPane extends StackPane
{
    String lbls[]=
    {"Yoda", "Obi-Wan Kenobi", "Luke Skywalker", "Mace Windu", "Darth Vader"};
    Button bs;
    void myHandleAction(ActionEvent e)
    {
        Button btn=(Button)e.getSource();
        System.out.println(btn.getText());
        this.getChildren().remove(btn);
        this.getChildren().add(0, btn);
    }
    public FXStackPane()
    {
        for(int i=0; i<lbls.length; i++)
        {
            bs=new Button(lbls[i]); bs.setMaxWidth(9999);
            bs.setOnAction(this::myHandleAction);
            getChildren().add(bs);
        }
    }
}
```



Click->



## Eventhandling via Method Reference

<http://code.makery.ch/blog/javafx-8-event-handling-examples/>

<https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html>



# JavaFX

## TilePane

- Entspricht einem Panel mit GridLayout.
- Alle Zellen haben gleiche Größe.
- Mit `setPrefColumns(n)` werden `n` Spalten definiert.
- Beispielsweise lässt sich das Tastenfeld des Taschenrechners mit einem `TilePane` bauen.



BorderPane mit Top  
und Center

TilePane mit 5 Spalten

# JavaFX

```
public class FXTilePane extends TilePane
{
    String lbls[]={"Yoda","Obi-Wan Kenobi", . . . ,"Darth Vader"};
    Button bs;

    void myHandleAction(ActionEvent e)
    {
        Button btn=(Button)e.getSource();
        System.out.println(btn.getText());
        this.getChildren().remove(btn); // aktueller Button entfernt
        this.getChildren().add(0,btn); // vorn neu eingefügt
    }
}
```

```
public FXTilePane()
{
    setPrefColumns(2);
    List<Node> l=getChildren();
    for(int i=0; i<lbls.length;i++)
    {
        bs=new Button(lbls[i]); bs.setMaxWidth(9999);
        bs.setOnAction(e->myHandleAction(e));
        l.add(bs);
    }
}
```



Click auf  
Darth Vader



# JavaFX

## AnchorPane

- Es können mehrere Nodes verwaltet werden.
- Zu jedem Node können die Abstände zu den Rändern gesetzt werden.
- Mit den statischen Funktionen können die Ränder die ein Node haben soll eingestellt werden:

```
setTopAnchor(b1, 5.0);  
setLeftAnchor(b1, 10.0);  
setRightAnchor(b1, 150.0);  
setBottomAnchor(b1, 5.0);
```

- Die einzelnen Nodes können sich dabei (teilweise) überdecken

# JavaFX

```
import javafx.scene.layout.*;
import javafx.scene.control.*;
import javafx.event.*;
```



```
public class FXAnchorPane extends AnchorPane
```

```
{
    Button b1=new Button("Miss Marple");
    Button b2=new Button("Mr. Stringer");
```

```
public FXAnchorPane()
{
```

```
    setTopAnchor(b1, 5.0);
    setLeftAnchor(b1, 10.0);
    setRightAnchor(b1, 150.0);
    setBottomAnchor(b1, 5.0);
    setTopAnchor(b2, 20.0);
    setRightAnchor(b2, 10.0);
    setBottomAnchor(b1, 20.0);
    getChildren().addAll(b1, b2);
```

```
    }
}
```

Es werden die Abstände zu den Rändern festgelegt.

Ohne diese Angabe liegen die Nodes u.Ust. übereinander

# JavaFX

## GridPane

- Entspricht in etwa dem GridbagLayout von AWT.
- Ist das mächtigste und variabelste Layout.
- Spannt ein Gitter von Zeilen und Spalten verschiedener Höhe und Breite auf.
- Den einzelnen Nodes werden die Anzeigeeigenschaften (Constraints) zugeordnet.
- Dafür steht eine große Anzahl statischer Methoden zur Verfügung
- Mit `myGrid.add(node, idxColumn, idxLine);` wird das Node node in das Grid in die angegebene Zeile/Spalte eingefügt.
- Nachfolgend wird schrittweise das Bauen des GUI gezeigt.



# JavaFX

Bau der Oberfläche im GridPane ohne Funktionalität

```
import javafx.event.*;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.stage.*;
import javafx.geometry.*;

class PocketCalc extends GridPane
{
    TextField tf=new TextField();
    String lbls[][]= {{ "M+", "7", "8", "9", "/" },
                    { "M-", "4", "5", "6", "*" },
                    { "MR", "1", "2", "3", "-" },
                    { "CE", "0", ".", "=", "+" } };
    . . .
}
```

# JavaFX

```
PocketCalc() // von GridPane abgeleitet!!
{
    tf.setEditable(false);
    tf.setPrefWidth(20);
    ColumnConstraints c=new ColumnConstraints();
    // einzige Moeglichkeit Insets zu setzen
    setConstraints(tf, 0, 0, 5, 1, Hpos.CENTER,
                  VPos.CENTER,Priority.ALWAYS ,Priority.ALWAYS ,
                  new Insets(5, 5, 5, 5));
    add(tf,0,0);
    for (int il=0; il<lbls.length; il++)
        for (int ic=0; ic<lbls[0].length; ic++)
        {
            Button b=new Button(lbls[il][ic]);
            b.setMaxWidth(999.9); b.setMaxHeight(999.9);
            setConstraints(b, ic, il+1, 1, 1, Hpos.CENTER,
                          Vpos.CENTER, Priority.ALWAYS, Priority.ALWAYS,
                          new Insets(5, 5, 5, 5));
            add(b,ic,il+1);
            connectToListeners(b,ic,il);
        }
}
```

# JavaFX

```
private void connectToListeners(Button b, int ic, int il)
{
    // Connect to listeners
    switch (ListenerIDs[il][ic])
    {
        // Lambda „expression“
        // case idcl: b.setOnAction(e->{tf.setText(""); . . .}); break;

        // Lambda calls function
        // case idcl: b.setOnAction(e-> {this::fcl();}); break;

        // using Function reference
        case idcl: b.setOnAction(this::fcl); break;
        case idnl: b.setOnAction(this::fnl); break;
        case iddl: b.setOnAction(this::fdl); break;
        case idca: b.setOnAction(this::fca); break;
        case idml: b.setOnAction(this::fml); break;
    }
}
```



# JavaFX

Dazu die Application class:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.*;

public class PocketCalcMain extends Application
{
    public static void main(String[] args)
    { launch(args); }

    PocketCalc p=new PocketCalc();

    @Override
    public void start(Stage primaryStage)
    {
        primaryStage.setTitle("JavaFX Caculator");
        Scene scene = new Scene(p/*, 300, 175*/);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

# JavaFX - fxml

- **Download:** <https://www.oracle.com/technetwork/java/javafxscenebuilder-1x-archive-2199384.html>
- Oberfläche wird mit xml beschrieben
- Oberfläche wird in der Starterclass generiert
- Oberfläche kann durch Änderungen des xml-Files modifiziert werden, ohne neu compilieren zu müssen
- Oberfläche kann mit JavaFXSceneBuilder1.1 erstellt („zusammengeclickt“) werden.
- Die Verbindung zum Java-Programm wird über die Eventhandler hergestellt, die via Funktionsreferenzen angebunden werden.
- Die Eventhandler sind in einer ControllerClass untergebracht, eintragen im Scenebuilder ganz untern Rechts (Controller)

# JavaFX - fxml

The screenshot shows the JavaFX IDE interface for editing the `FXMLPocketCalc.fxml` file. The main workspace displays a calculator UI with a grid of buttons. The left sidebar is divided into a Library and a Controller section. The Library shows various components like `Accordion`, `AnchorPane`, and `ButtonBar`. The Controller section shows the `FXMLPocketCalcController` class selected, with a table listing assigned `fx:id` values and their corresponding components.

fx:id	Component
tf	TextField

The right sidebar shows the Inspector for the selected `TextField` component, with the `fx:id` property set to `tf`. The Inspector also shows various event handlers for the component.

Hier  
Controller  
eintragen

# JavaFX - fxml

- Scenebuilder:

- Download von Oracle

<http://www.oracle.com/technetwork/java/javafxscenebuilder-1x-archive-2199384.html>

- Kann verwendet werden mit

- Netbeans
- Eclipse
- standalone mit bel. Editor

- Aufruf unter Ubuntu:

```
/opt/SceneBuilder/SceneBuilder
```

# JavaFX - fxml

```
public class JavaFXMLPocketCalc extends Application {  
  
    @Override  
    public void start(Stage stage) throws Exception {  
        Parent root =  
            FXMLLoader.load(getClass().  
                getResource("FXMLPocketCalcTile.fxml"));  
  
        Scene scene = new Scene(root);  
  
        stage.setScene(scene);  
        stage.show();  
    }  
  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
        launch(args);  
    }  
}
```

# JavaFX - fxml

```
public class FXMLPocketCalcController implements Initializable
{
    private boolean clear=false;
    private boolean dot =false;
    private double mem=0.0;
    private double value[]={0.0, 0.0};
    private char op[]={0,0};
    @FXML

    private TextField tf;

    @FXML
    public void handleButtonActionNL(ActionEvent e)
    {
        Button x=(Button)e.getSource();
        String s=x.getText();
        if (clear)
        {
            tf.setText(s);
            dot =false;
            clear=false;
        }
        else      tf.appendText(s);
    }
}
```

# JavaFX - fxml

```
public void handleButtonActionDL(ActionEvent e)
{
    if (!dot)tf.appendText(".");
    dot=true;
}
public void handleButtonActionML(ActionEvent e)
{
    Button x=(Button)e.getSource();
    String s=x.getText();
    switch(s.charAt(1))
    {
        case 'R': tf.setText(""+mem)          ;break;
        case '+': mem+=Double.parseDouble(tf.getText()) ;break;
        case '-': mem-=Double.parseDouble(tf.getText()) ;break;
    }
}
public void handleButtonActionQuit(ActionEvent e)
{
    System.exit(0);
}
@Override
public void initialize(URL url, ResourceBundle rb) {
    // TODO
}
```

# JavaFX - fxml

The screenshot shows the JavaFX Scene Builder interface for a file named 'FXMLPocketCalc.fxml'. The main workspace displays a calculator UI on a 5x5 grid. A yellow callout bubble points to the 'On Action' property in the Inspector panel, which is set to '# handleButtonActionNL'. The Inspector panel also shows 'Layout' and 'Code' sections for the selected Button. The Hierarchy panel on the left shows the structure of the UI, including a GridPane with a TextField and several Buttons.

**Library** Search  GridPane : 5 x 5 Button : 5

**Containers**

- Accordion
- Anchor Pane
- Border Pane
- Flow Pane
- Grid Pane
- HBox
- Pane
- ScrollPane
- Split Pane (Horizontal Flow)
- Split Pane (Vertical Flow)
- Stack Pane

**Hierarchy**

- GridPane 5 x 5
  - TextField (0, 0)
  - Button (0, 1) M+
  - Button (1, 1) 7
  - Button (2, 1) 8
  - Button (3, 1) 9
  - Button (4, 1) /
  - Button (0, 2) M-
  - Button (1, 2) 4
  - Button (2, 2) 5
  - Button (3, 2) 6
  - Button (4, 2) \*
  - Button (0, 3) MR
  - Button (1, 3) 1
  - Button (2, 3) 2
  - Button (3, 3) 3
  - Button (4, 3) -
  - Button (0, 4) CE
  - Button (1, 4) .
  - Button (2, 4) 0
  - Button (3, 4) =
  - Button (4, 4) +

**Inspector** Search

**Properties** : Button

**Layout** : Button

**Code** : Button

fx:id

On Action  
#

**Drag and Drop**

On Drag Detected  
#

On Drag Done  
#

On Drag Dropped  
#

On Drag Entered  
#

On Drag Exited  
#

On Drag Over  
#

On Mouse Drag Entered  
#

Hier wird die CallbackFunktion als Eventhandler eingetragen