JAVA FX

JAVA FX

<> Übersicht

<> Architektur

<> UI Elemente

<> CSS Box Modell

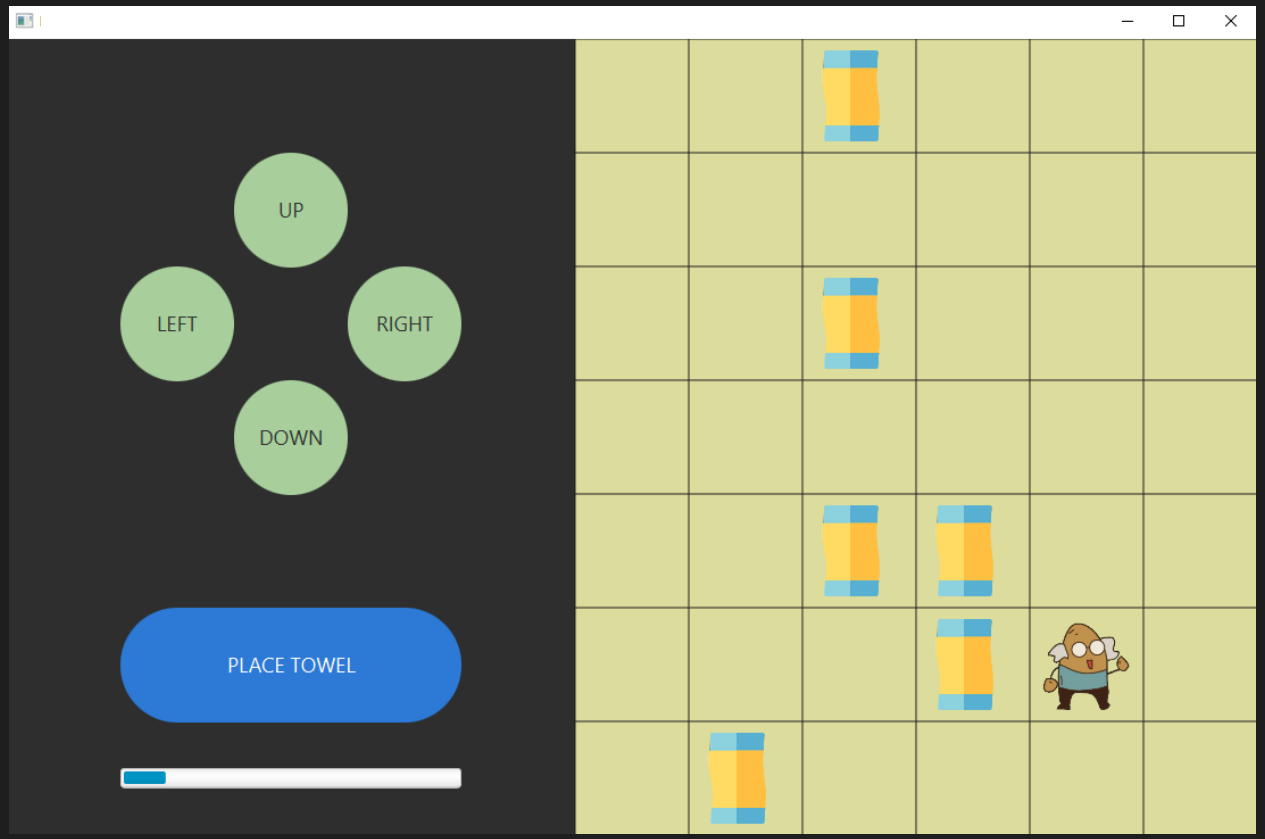
<> Instanziierung während der Laufzeit (Übung 1)

<> Erstellung vor der Laufzeit (Übung 2)

<> Data Binding (Übung 3)

<> Exkurs: MVC (optional)

<> Events (Übung 4 optional)



Entwicklung und Vorteile

<> von Oracle geplant als Ablösung von Swing und AWT (Abstract Window Toolkit), Einführung 2008

<> ermöglicht modernere und dem OS angepasste Optik – weg von „Java-Style“

<> struktureller Aufbau entspricht etablierten Webstandards:
- <> Layout und Deklaration der Elemente in Markup
- <> Nutzung von Stylesheets
- <> entkoppelte Code-Logik

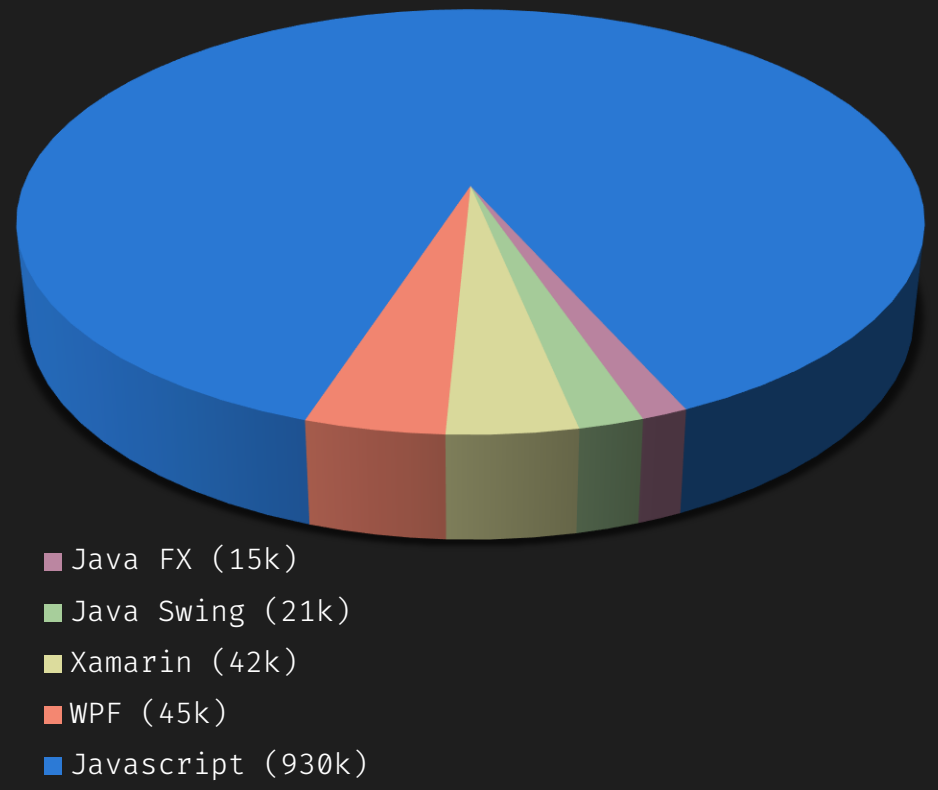
<> konzeptionell übertragbar auf andere GUI-Frameworks (WPF, Xamarin, XAML, PySimpleGUI)

<> mit JavaPorts und JavaFX Mobile auch mobile Unterstützung

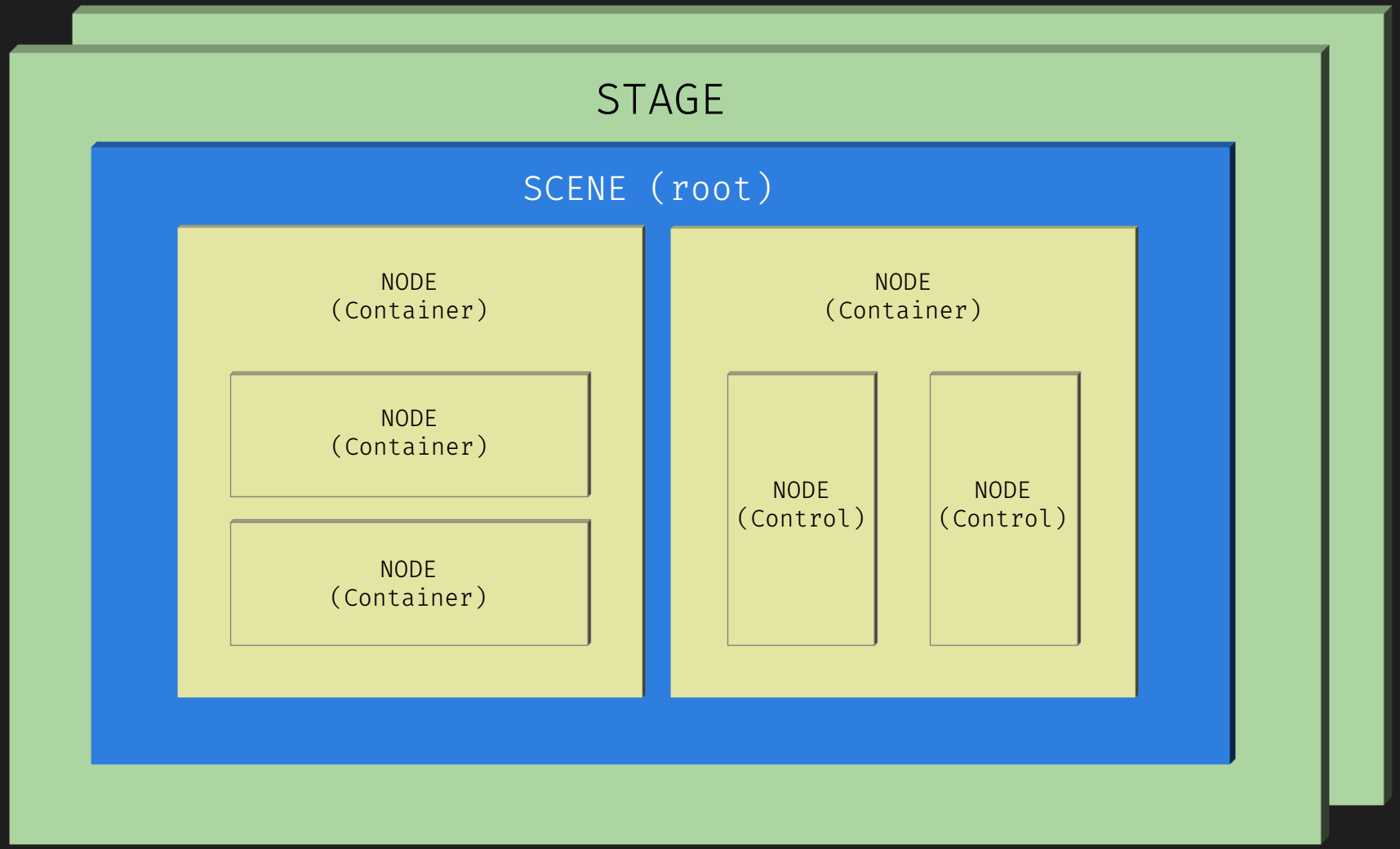
<> Webview ermöglicht einige HTML5 Elemente (z.B. JS Wrapper, Forms, Mediaelements)

→ trotzdem geringe Relevanz (vs.: Android Studio, JS und immer noch Swing), Java ist und bleibt eher Backend-Sprache

GitHub Repos (bei Suche nach entsprechendem Begriff)
Stand 5/21)



Architektur

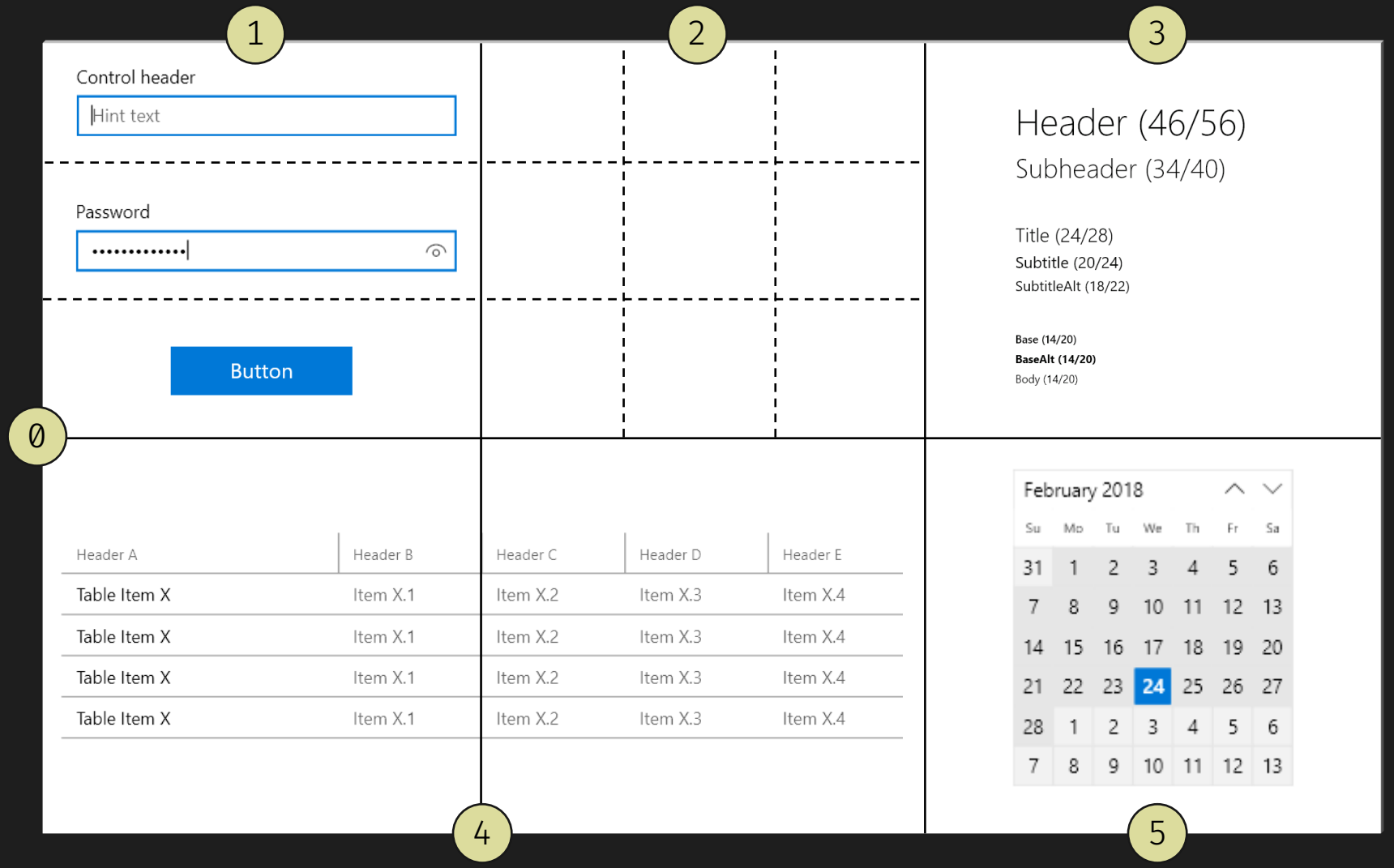


Stage:
<> Programmfenster
<> Programmeinstieg
<> Parameter in Main

Scene:
<> Verwaltung der Nodes in hierarchischer Treestruktur (Scene selbst ist der root)
<> können in einer Stage „gewechselt“ werden
<> Initialisierung von Controllerklasse

Nodes:
<> einzelne GUI-Elemente
<> Layout und Control
<> Childs der Scene und ggfs. selbst Parentnodes
<> Zuweisung von IDs
<> Registrierung von Event-Listenern

Layout und Controls



0
3*2 Grid

1
Text Input
Buttons
Vbox

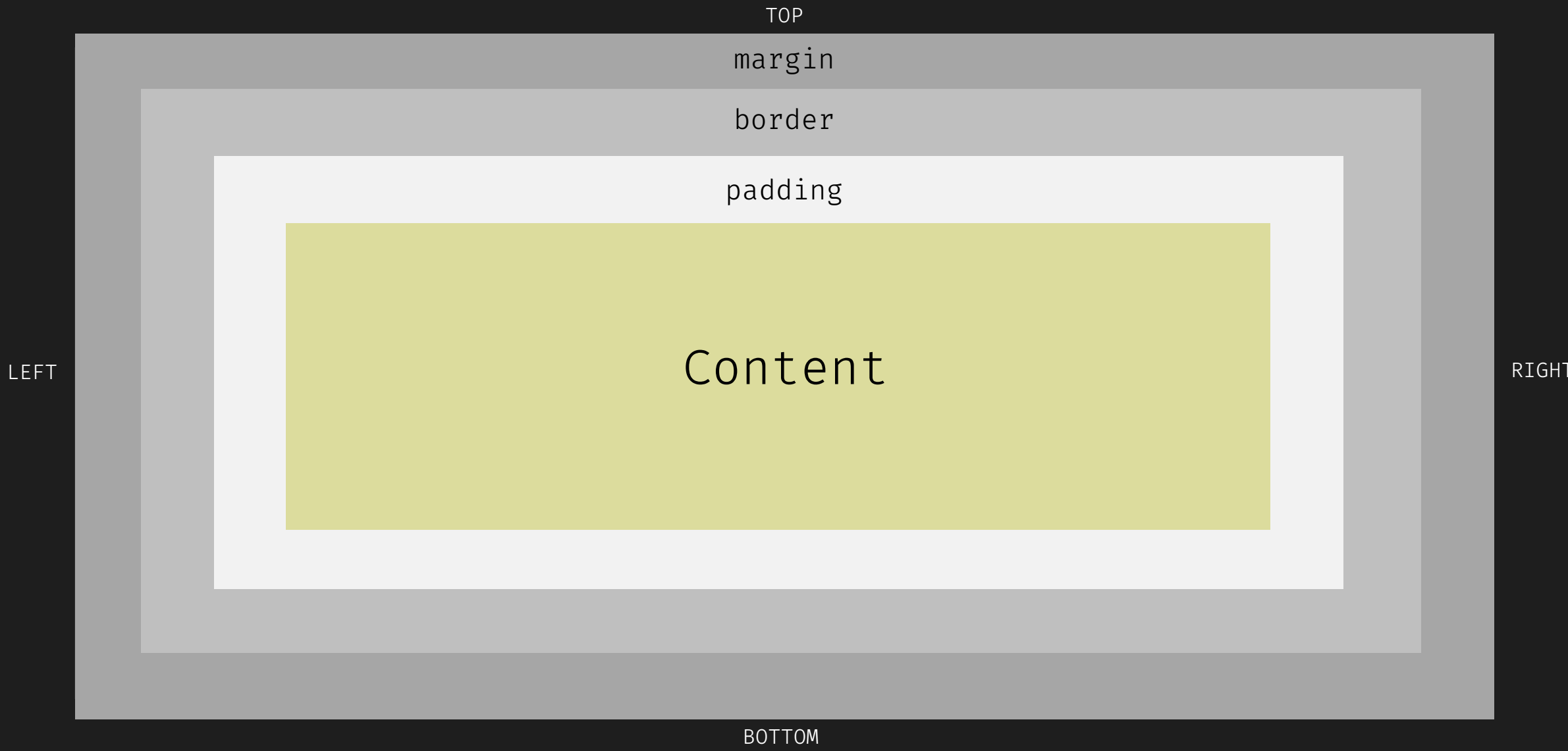
2
3*3 Grid

3
Text

4
Table (Columnspan 2)

5
Sonstige

CSS - Box Modell



CSS - Box Modell (Forts.)

Instanziierung während der Laufzeit

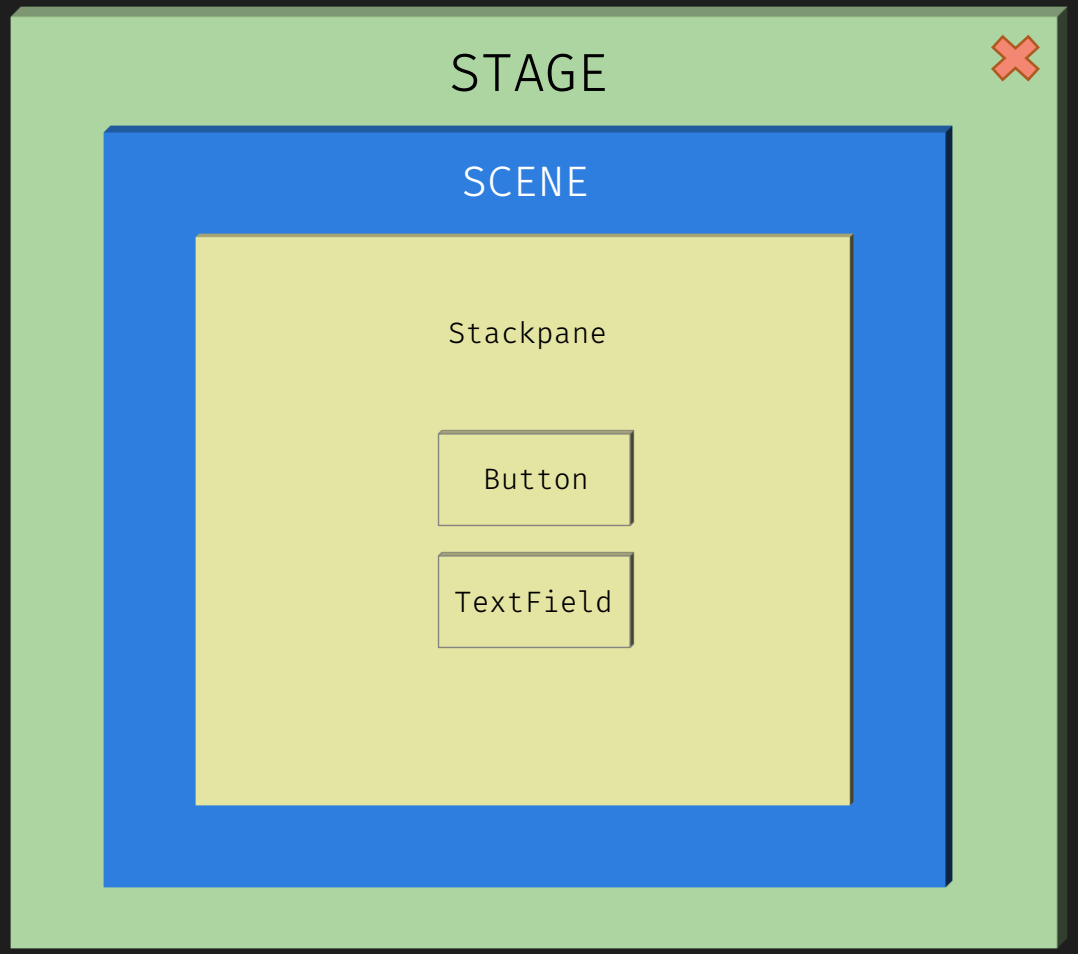
```java
@Override
public void start(Stage primaryStage)
{
    StackPane root = new StackPane();
    Button btnIncrement = new Button();
    Text txtIncrement = new  Text();

    StackPane.setMargin(txtIncrement, new Insets(50,0,0,0));
    btnIncrement.setStyle("-fx-background-color: #A7CE9B;");
    btnIncrement.setText("+1");

    btnIncrement.setOnAction(new EventHandler<ActionEvent>()
    {

        @Override
        public void handle(ActionEvent event)
        {
            incrementThis++;
            txtIncrement.setText(String.valueOf(incrementThis));
        }
    });

    root.getChildren().addAll(btnIncrement, txtIncrement);
    primaryStage.setScene(new Scene(root, 300, 250));
    primaryStage.show();
}
```



Häufig verwendete Methoden

Methode	Beispiel Aufruf	Typ (Parameter bzw. Rückgabe)	Klasse
setText(...)	btnIncrement.setText("+1");	String	Text, Textboxen, Buttons
getText()	btnIncrement.getText();	String	Text, Textboxen, Buttons
setStyle(...)	btnIncrement.setStyle("-fx-background-color: #A7CE9B;");	String der einer CSS Klasse entsprechen muss	alle
setMargin(...)	StackPane.setMargin(txtIncrement, new Insets(50,0,0,0));	Zu änderndes Objekt, Marginwerte	Containerklassen (statischer Aufruf)
getChildren()	root.getChildren().addAll(btnIncrement, txtIncrement);	Liste von Nodes, muss aufgerufen werden bevor z.B. die add Methode benutzt wird (s. Beispiel)	alle die Kinderobjekte enthalten können
setScene(...)	primaryStage.setScene(new Scene(root, 300, 250));	Scene-Object	Stage
show()	primaryStage.show();	muss aufgerufen werden, damit das Hauptfenster gerendert wird	Stage
bind(...)	txtShowInput.textProperty().bind(propertyToBind)	zu bindende Property, Aufruf von Property, nicht direkt aus Datentyp (s. links)	alle, mit denen mit Hilfe der Methode ...Property() diese ausgelesen werden können

Häufig verwendete Methoden (Forts.)

Methode	Beispiel Aufruf	Typ (Parameter bzw. Rückgabe)	Klasse
initialize()	@FXML initialize(){...}	Void, überschreibt die init der Controllerklasse	jeweilige Controllerklasse
getSource()	var button = (Button)actionEvent.getSource();	Event Quelle	ActionEvent
getKeyCode()	var keyCode = key.getCode(); if (keyCode == KeyCode.ENTER){...}	KeyCode (Tastaturkey)	KeyEvent
add(...)	playGrid.add(imageObject, columns, row);	Objekt welches hinzugefügt werden soll, Spalte, Zeile	Layoutelemente (z.B. Grid)
remove(...)	playGrid.getChildren().remove(objectToRemove);	Kinderobjekt welches entfernt werden soll	alle die Kinderobjekte enthalten können
setDisable(...)	nodeObject.setDisable(true);	boolean	fast alle Objekte
setVisible(...)	nodeObject.setVisible(false);	boolean	fast alle Objekte
setMinSize(...) setMaxSize(...)	nodeObject.setMinSize(100);	double	Fast alle Objekte

Challenge #1

```
var challengeOne = new Dictionary<Nummer, Vorschlag>()
{
    1. : Ändert die Farbe des Buttons
    2. : Ändert die Größe des Buttons
    3. : Ändert den Abstand des Textes zum Buttons
    4: : Entfernt den Button nach 10 Klicks
    5. : Fügt einen zweiten Button hinzu, der den Zähler dekrementiert
};

// Tip: Checkt die Methoden des Buttonobjektes
// Ordner: JavaFX PostLayout
```

Vordefinierte Struktur und Aufbau im Code

Stage:
<> Aufruf in der Main
<> Laden der (F)XML
<> Laden der CSS
<> .show()

Scene:
<> (F)XML kann Elemente definieren
<> Nodezugriff in Controllerklasse

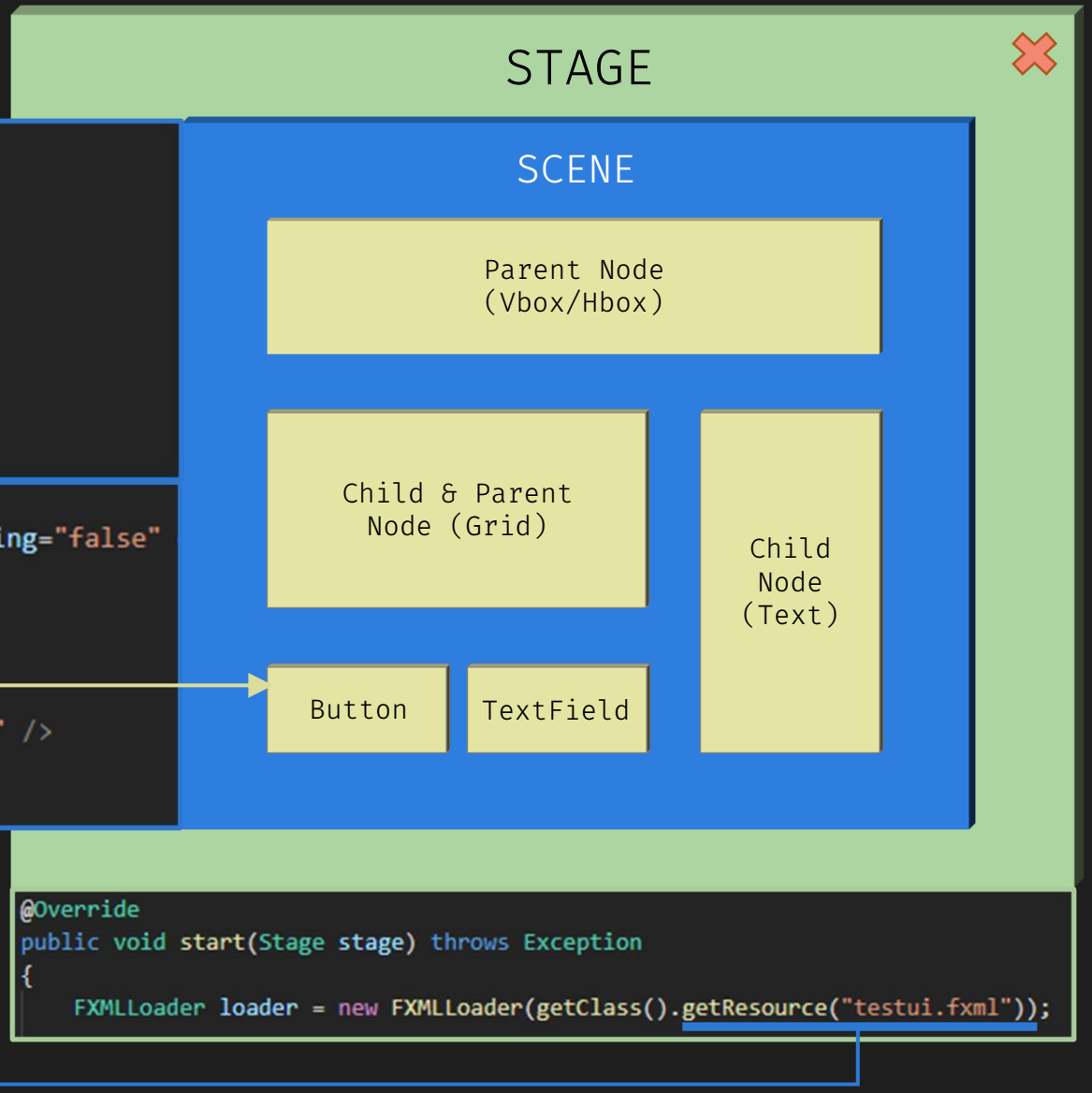
```
public class ThisSceneController
{

    @FXML
    private Text textOutputField;
    @FXML
    private Button buttonOne;
    @FXML
    private TextField userInput;
```

```
<children>
    <Button fx:id="buttonOne" mnemonicParsing="false"
        <GridPane.margin>
            <Insets />
        </GridPane.margin>
        <font>
            <Font name="Consolas" size="20.0" />
        </font>
    </Button>
```

```
.button {
    background-color: red;
    -fx-background-color: red;
    -fx-text-fill: white;
}
```

```
@Override
public void start(Stage stage) throws Exception
{
    FXMLLoader loader = new FXMLLoader(getClass().getResource("testui.fxml"));
```



Scene Builder

<> WYSIWYG – basierte Lösungen zur Erstellung der XML Files und Generierung der Objekte

<> nutzt unbedingt Containerobjekte für einen sauberen Aufbau des Layouts!

<> Drag and Drop direkt in die Stage vermeiden!

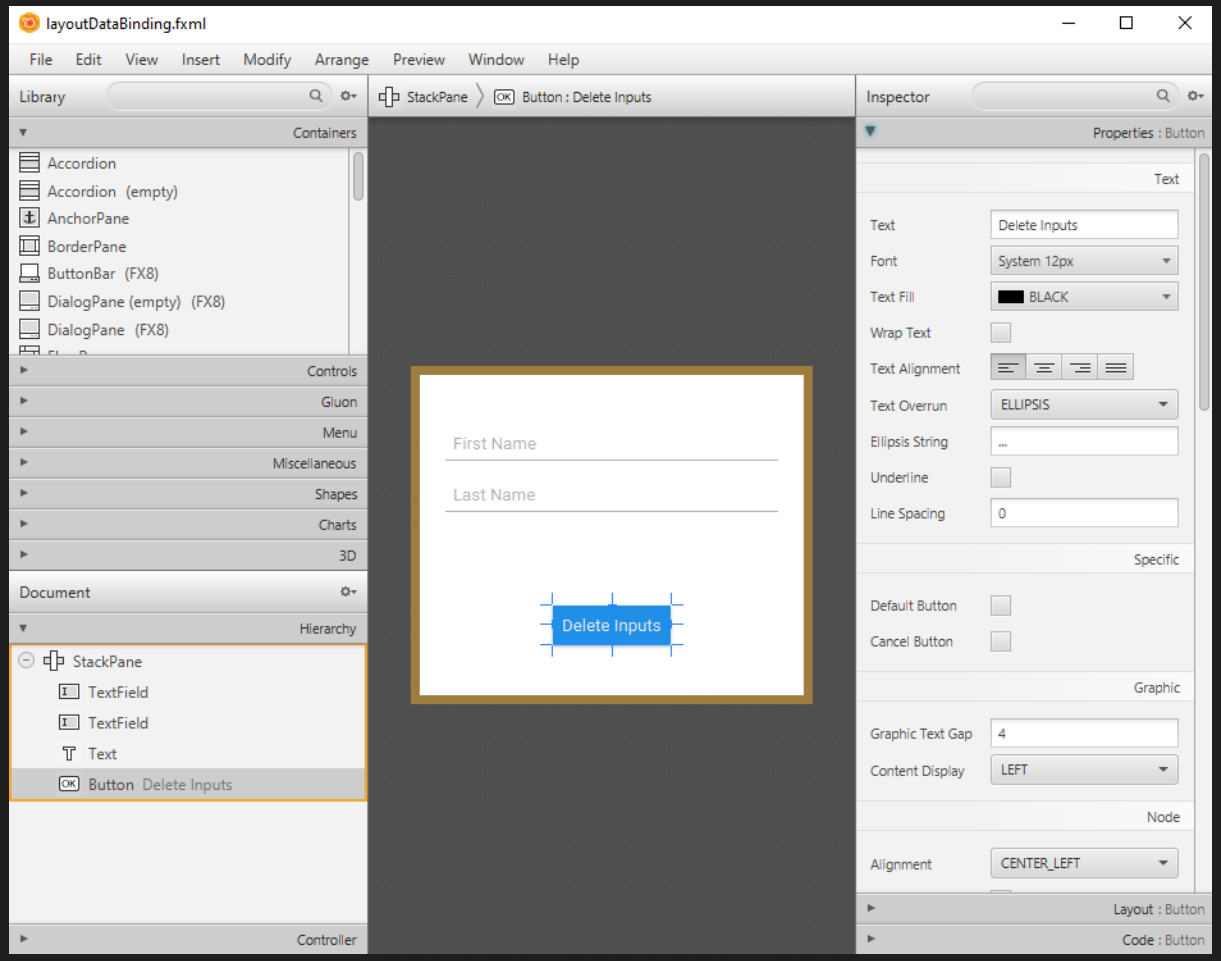
<> Vorschaufunktion

<> Definition der Ids (Konvention)

<> Definition der Eventhandler

<> Definition der Controllerklasse (nützlich: view -> show sample controller skelleton)

<> Oracle Scene Builder, Gluon Scene Builder, teilweise integriert in IDE (eclipse?)

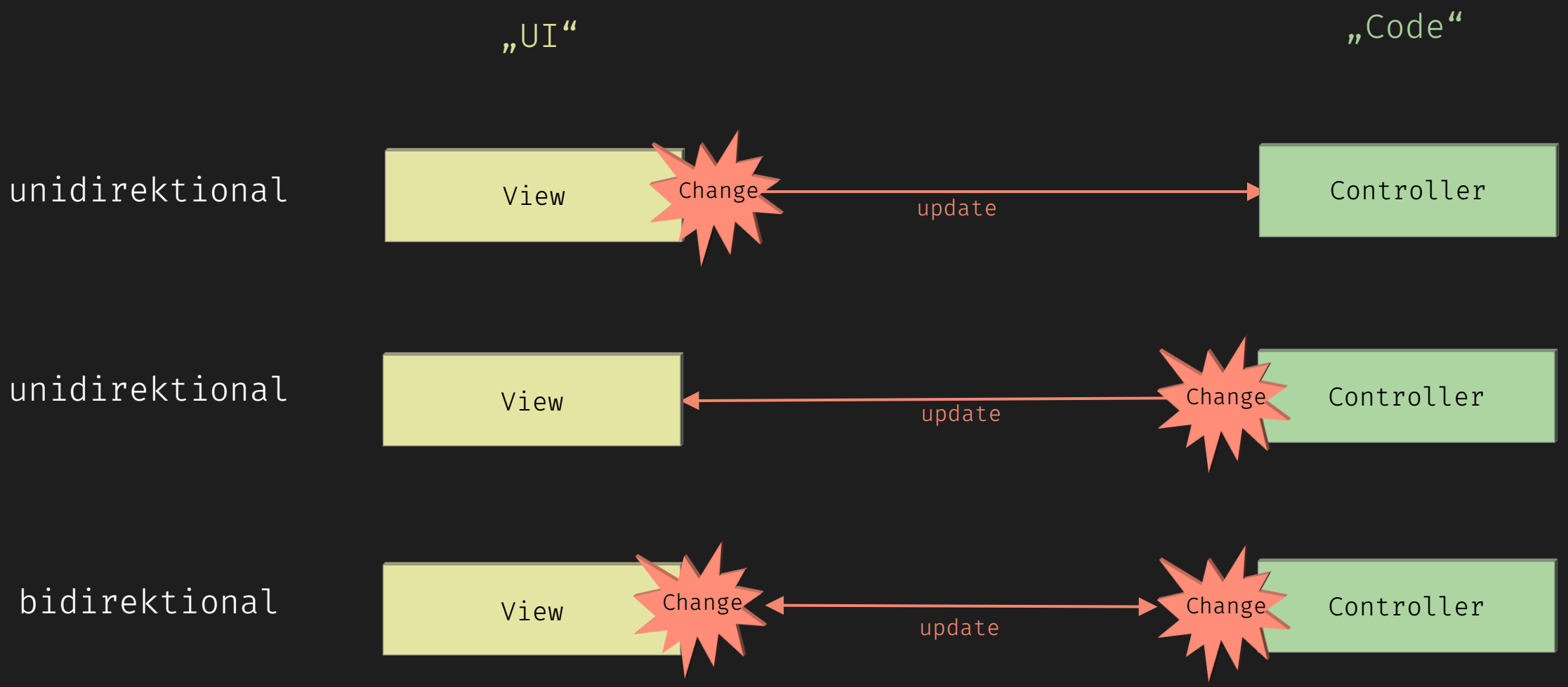


Challenge #2

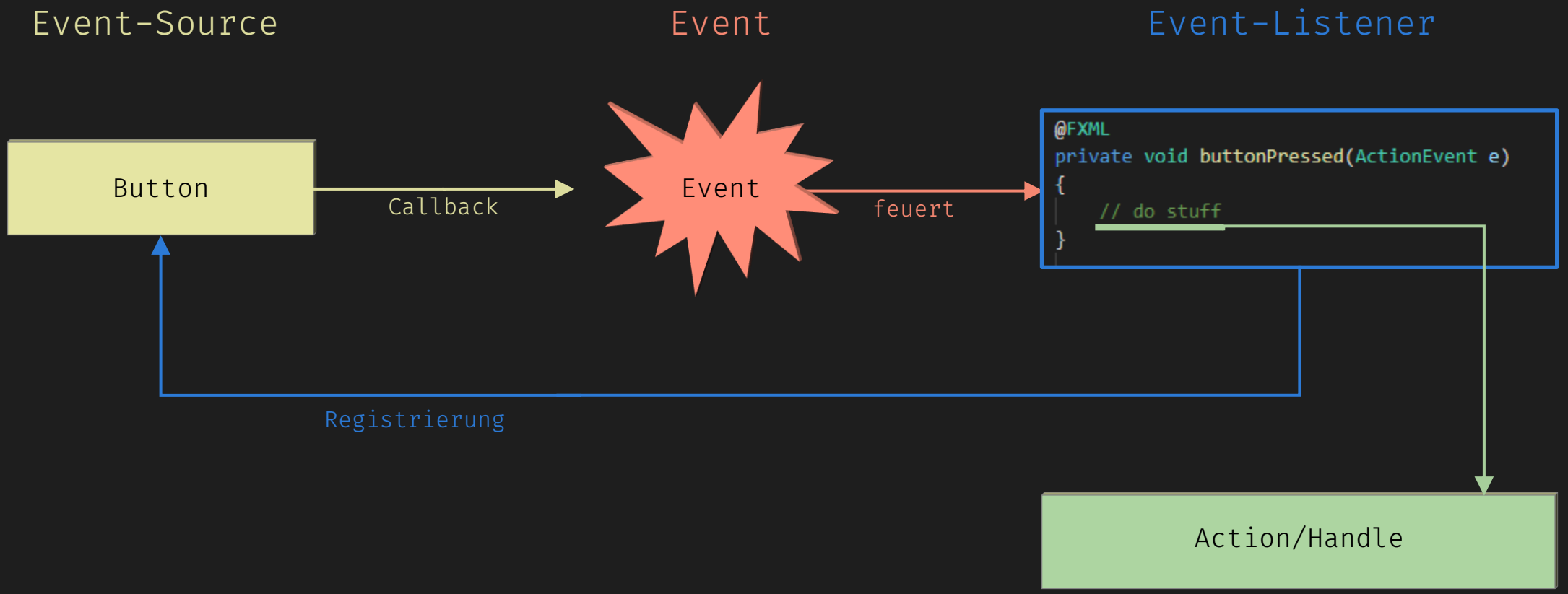
```
var challengeTwo = new Dictionary<Nummer, Vorschlag>()
{
    1. : Fügt im Scene Builder einen zweiten Button hinzu
    2. : Weist diesem Button einen zweiten Eventlistener zu
    3. : Manipuliert im zweiten Listener den Incrementwert (egal wie)
};

// Tip: Öffnet die Datei layoutExample.fxml mit Gluon Scene Builder
// Ordner: JavaFX PreLayout
```

Data-Binding



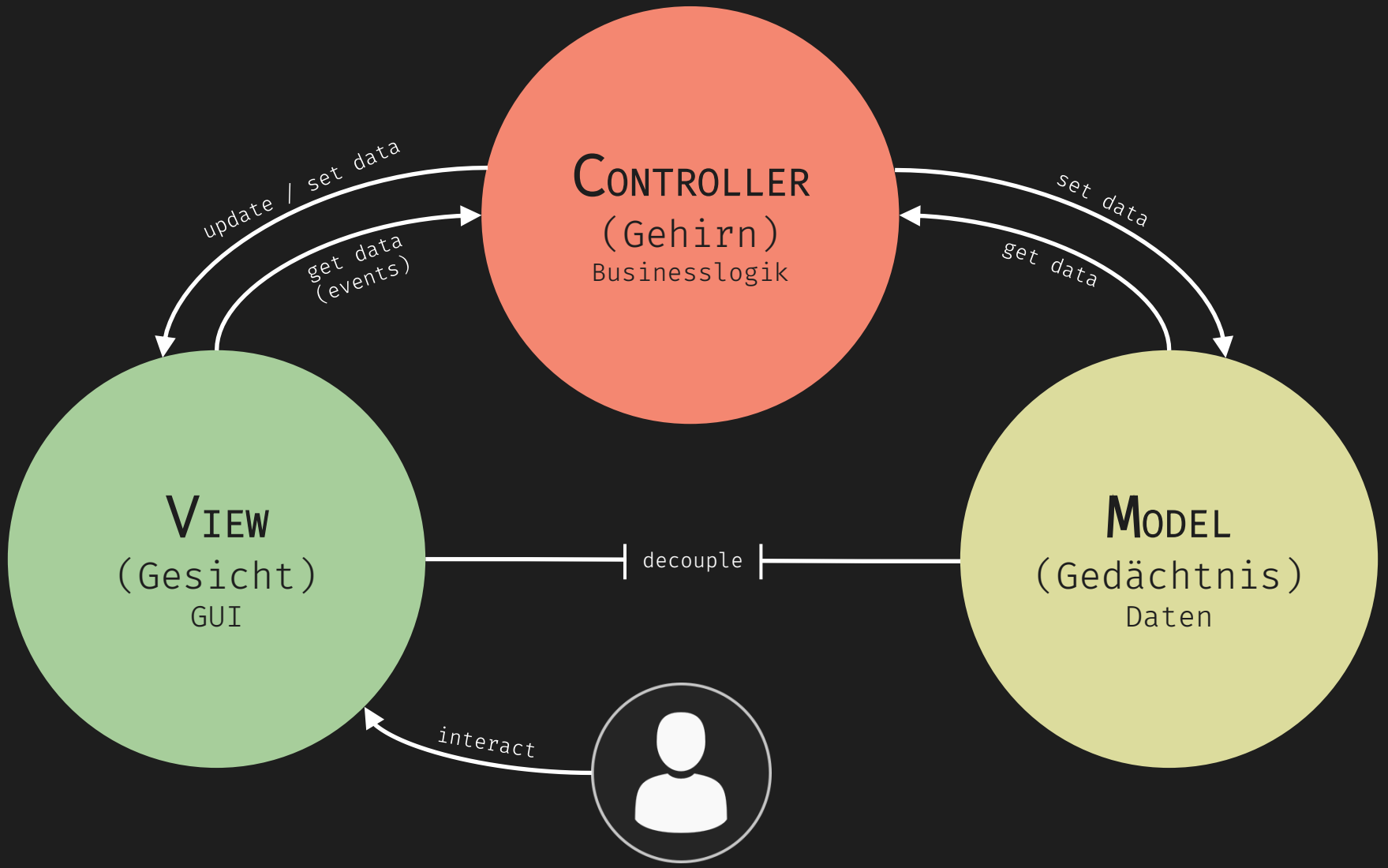
Events


Event-Source
Event
Event-Listener
Button
Callback
Event
feuert
@FXML
private void buttonPressed(ActionEvent e)
{
// do stuff
}
Registrierung
Action/Handle

Challenge #3

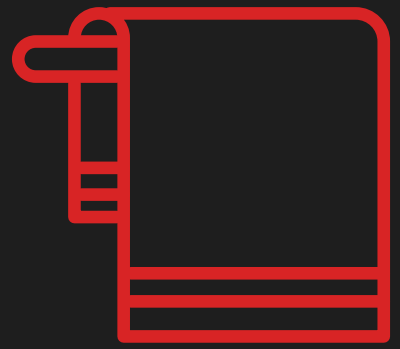
```
var challengeThree = new Dictionary<Nummer, Vorschlag>()
{
    1. : Fügt ein weiteres Element hinzu, welches Text beinhalten kann
    2. : Bindet den Inhalt des vorhandenen Textfeldes an das neue Textfeldes
    3. : Fügt ein weiteres Eingabefeld hinzu
    4. : Fügt einen weiteren Button hinzu, der beim Klick die Bindung vom
         ersten Eingabefeld an das Textfeld aufhebt und eine Bindung vom
         neuen Eingabefeld an das Textfeld ändert. Ein erneuter Klick stellt
         die alte Bindung wieder her.
}
```

Exkurs: MVC


update / set data
get data (events)
Controller
(Gehirn)
Businesslogik
set data
get data
View
(Gesicht)
GUI
decouple
Model
(Gedächtnis)
Daten
interact

Challenge #4

```
var challengeThree = new Dictionary<Nummer, Vorschlag>()
{
    1. : Experimentiert mit dem "Douglas Adams" Programm
    2. : Nutzt möglichst viele Techniken und Elemente
         (Events, Databinding, UI Elemente, Erstellung vor und in
         der Laufzeit, CSS Styling)
}
```



„DON‘T PANIC
AND CARRY A TOWEL“