GUI

SS 2019

Grafische Oberflächen

- Oberfläche
- Steuerungslogik
- Anwendungslogik

Oberfläche*

- Fenster mit weiteren Komponenten
  - Realisierung
    - Klasse, die von JFrame erbt
      - `public class GUI extends JFrame { … }`
    - Layoutmanager: Methoden zum Setzen eines Layouts
      - `setLayout(new FlowLayout())`
      - …
    - Im Konstruktor werden die einzelnen Komponenten hinzugefügt
      - Wie? Das hängt vom Layout ab, z.B.:
        - `this.add(component)` bei FlowLayout
    - Methoden zum Verhalten beim Schließen des Fensters
      - `setDefaultCloseOperation(EXIT_ON_CLOSE)`
      - …

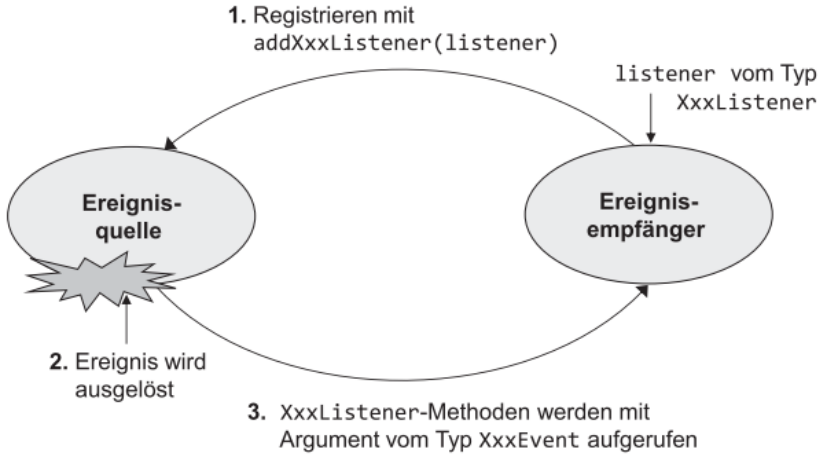
* Mit Swing

Layout Beispiele

- Border 
- Flow 
- Grid 
- Hbox 
- ...

Steuerung

- Verhalten hinzufügen
  - Verbindung der GUI-Komponenten mit „Listener“



- Ereignisbehandlung:
  - Listener „lauschen“ auf bestimmte Ereignisse, die von einer GUI-Komponente ausgelöst werden (->Registrieren)
  - eine GUI-Komponente löst ein Ereignis aus
  - Listener reagieren auf Ereignisse

[DA]

Ereignisbehandlung

- Realisierung:
  - eine GUI-Komponente löst ein Ereignis aus
    - Ereignisse sind in Java in Form von Event-Klassen implementiert (Paket java.awt.event enthält Event-Klassen, z.B. `ActionEvent`)
    - zum Beispiel Klicken auf ein Button => eine Instanz von ActionEvent wird erstellt
  - Listener „lauschen" auf bestimmte Ereignisse, die von einer GUI-Komponente ausgelöst werden
    - Weil vorher an der GUI-Komponente ein Listener registriert wurde
  - Listener reagieren auf Ereignisse
    - zum genannten `ActionEvent` gibt es in `java.awt.event` zum Beispiel das Listener-Interface `ActionListener`

Listener

- Sind Interfaces
  - die eine oder mehr Methoden (`actionPerformed()`) haben, die auf verschiedene Events reagieren
  - deren Methoden man implementiert
  - deren Methoden als Parameter eine Instanz des Events (`ActionEvent`) bekommen
  - Von denen ein Listener-Objekt erzeugt wird, das über entsprechende zur Verfügung stehende Methoden der GUI-Komponente hinzugefügt wird
    - addActionListener()

Listener Beispiele

- ActionListener
  - Z.B. bei Drücken eines Buttons
- TextListener
  - Bei Eingaben in Textfelder
- MouseListener
  - bei Drücken, Loslassen der Maustasten, …
- KeyListener
  - Abfangen von Tastatureingaben

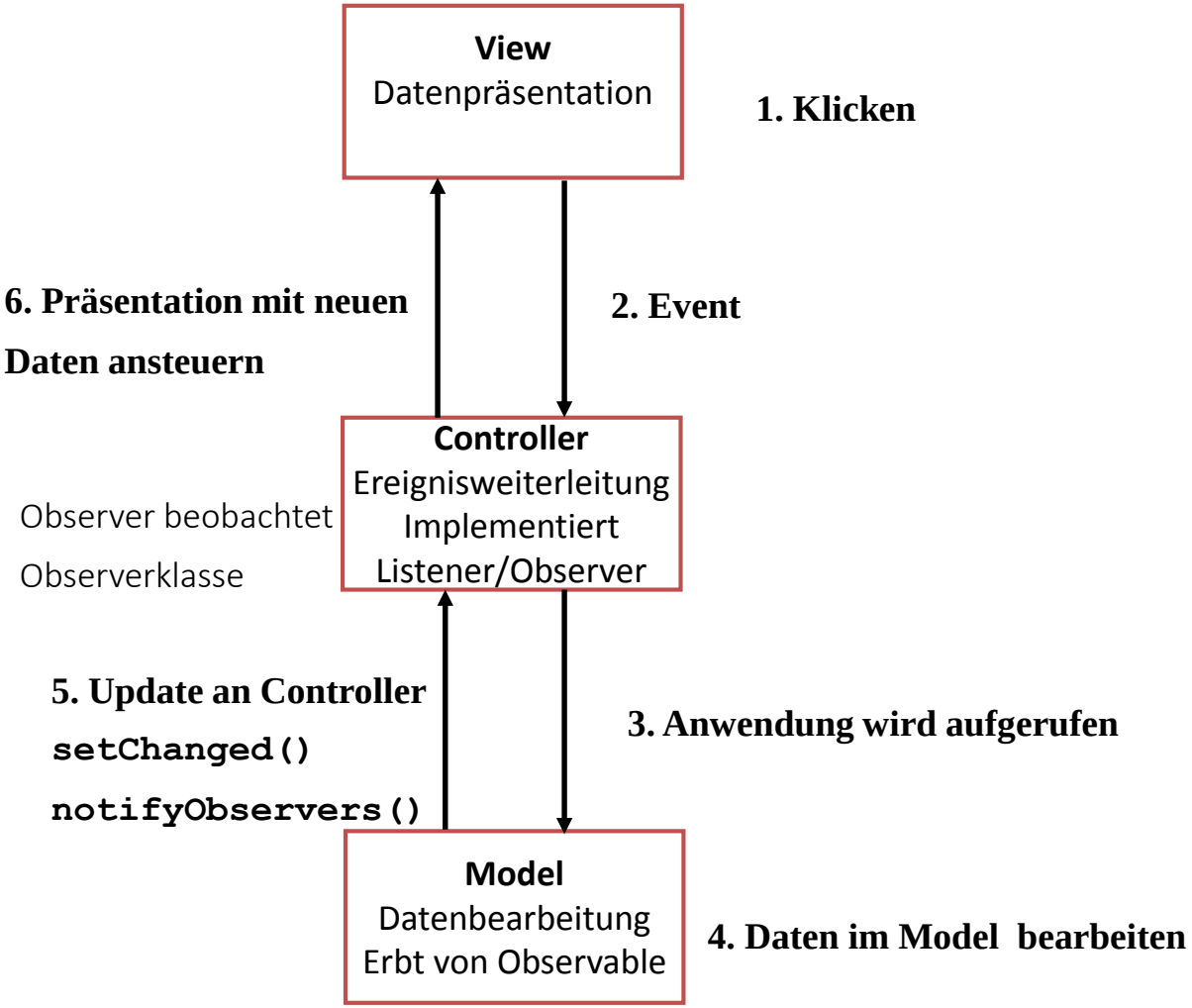
Anwendungslogik

- Die eigentliche Anwendung
  - Verwalten von Daten
  - Berechnen von Daten
  - …
- Methoden werden von Steuerung aufgerufen

Motivation MVC

- Trennung Oberfläche/Steuerung/Anwendungslogik
- Insbesondere
  - Lose Bindung zwischen
    - View (Oberfläche)
    - Model (Anwendung)
  - Mehrere Views für ein Model
- Bindung über Steuerung
- MVC = Model-View-Controller
  - Model ist die Anwendungslogik
  - View ist die Präsentation des Models (z.B. graf. Oberfläche)
  - Controller reagiert auf Eingaben von View und gibt sie an das Model weiter

MVC

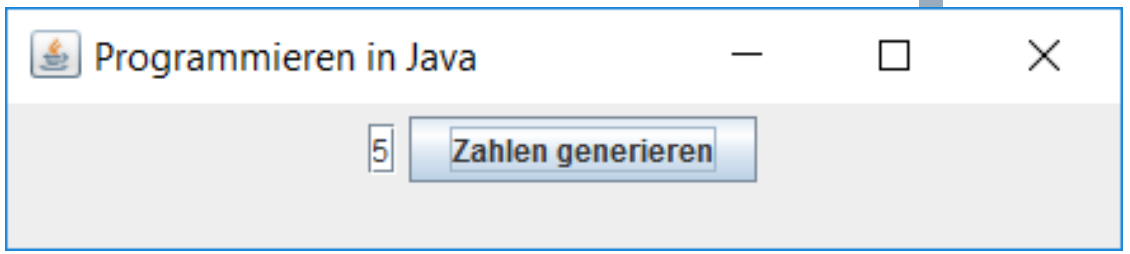


Beispiel

```
2  public class Main {
3
4      public static void main(String[] args) {
5          Model model = new Model();
6          ViewGUI viewGUI = new ViewGUI();
7
8          ControllerGUI controllerGUI = new ControllerGUI(model, viewGUI);
9
10     }
11
12 }
```

```
1  import java.util.Observable;
2
3  public class Model extends Observable {
4
5      private int zahl = 0;
6
7      public void zaehlen(){
8          zahl=zahl+1;
9          setChanged();
10         notifyObservers();
11     }
12     public int getZahl() {
13         return zahl;
14     }
15 }
```

Beispiel




```java
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.JButton;

public class ViewGUI extends JFrame {

    private JTextField text = new JTextField();
    private JButton knopf = new JButton("Tue irgend was");

    public ViewGUI(){
        this.setTitle("Programmieren in Java");

        this.setDefaultCloseOperation(EXIT_ON_CLOSE);

        this.setLayout(new FlowLayout());

        this.add(text);

        knopf.setText("Zahlen generieren");
        this.add(knopf);

        this.pack();

    }

    public JTextField getText() {
        return text;
    }

    public JButton getKnopf() {
        return knopf;
    }

}
```

```java
import java.util.Observer;
import java.util.Observable;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ControllerGUI implements Observer, ActionListener {

    private Model model;
    private ViewGUI view;

    public ControllerGUI(Model model, ViewGUI view) {
        this.model = model;
        this.view = view;

        model.addObserver(this);

        view.getKnopf().addActionListener(this);
        view.getText().setText(String.valueOf(model.getZahl()));

        view.setVisible(true);
    }
    @Override
    public void update(Observable arg0, Object arg1) {
        view.getText().setText(String.valueOf(model.getZahl()));
    }

    @Override
    public void actionPerformed(ActionEvent ae) {
        model.zaehlen();
    }
}
```


JavaFX

- Fenster ist jetzt eine Theaterbühne -> Stage
- Inhalt dieser Bühne -> Repräsentation über Instanzen der Klasse Scene
- Innerhalb einer solchen Szene sind GUI-Komponenten angeordnet.
- Layout
  - Swing: Instanz des Layoutmanagers und Zuordnung des Layouts für das aktuelle JFrame
  - JavaFX: Innerhalb meiner Bühne definiert mein Pane das Layout

JavaFX

- Eine JavaFX-Anwendung leitet immer von javafx.application.Application ab
- dort ist die Methode launch(), die man aus einer main Methode aufrufen kann, um die JavaFX Anwendung zu starten.

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

JavaFX

- In der start() Methode wird GUI initialisiert

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

//private Komponenten
  @Override
  public void start(Stage buehne) throws Exception {
    buehne.setTitle(„mein Titel");
    //initialisiere Komponenten
    //ordne Komponenten an
    buehne.show();
  }
}
```

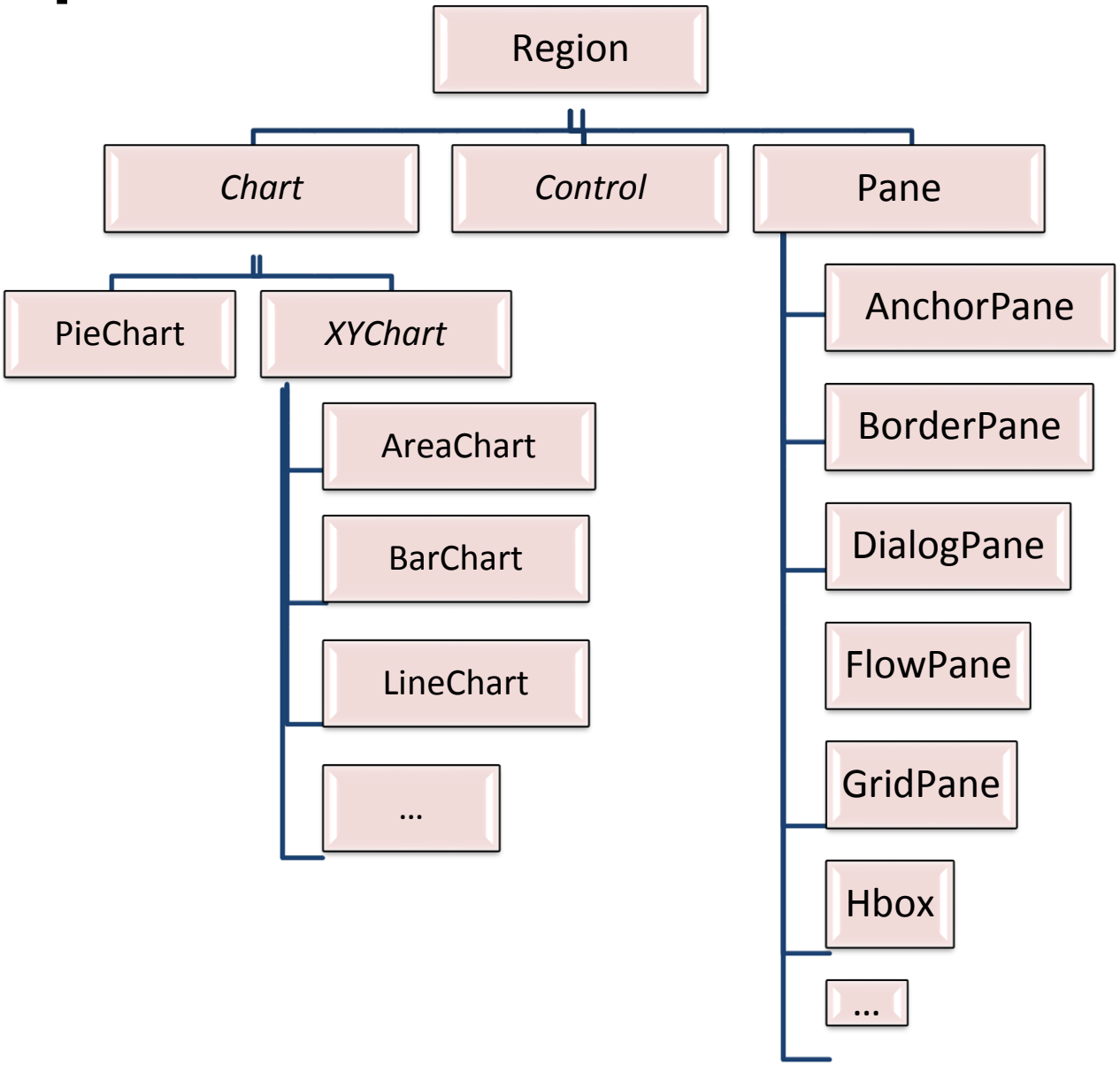
Szenegraph

- Der Inhalt einer Szene wird im Scenegraph definiert, der baumartig (hierarchisch) alle Grafikelemente enthält.
- Die Wurzel des Baumes gibt das Layout auf der obersten Ebene vor
  - StackPane, BorderPane, GridPane, FlowPane,
  - also z.B.: `BorderPane root = new BorderPane ();`
- Der Wurzel (root) werden die einzelnen Komponenten hinzugefügt.

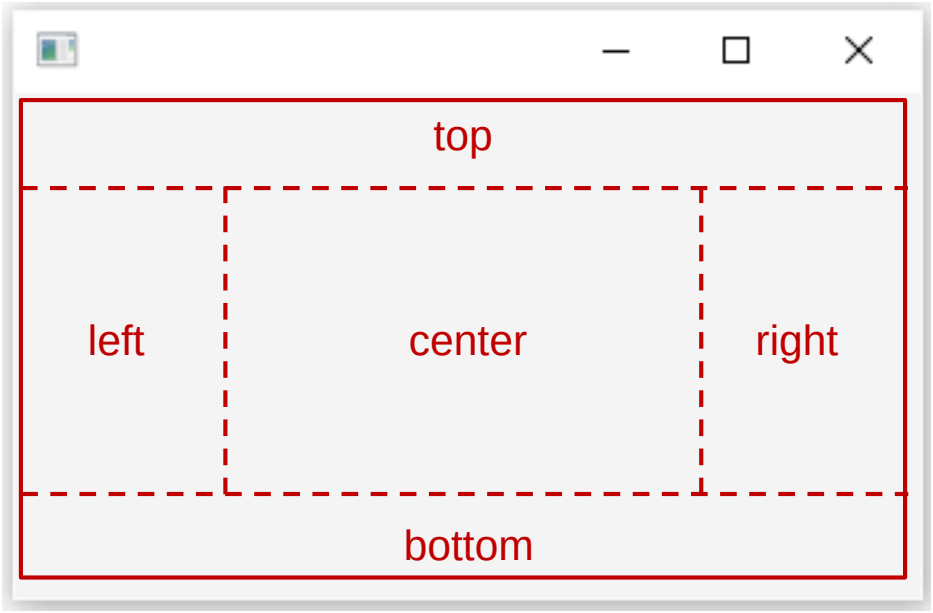
```
Label lbl = new Label(" Hello World");

root.getChildren().add(lbl);
```

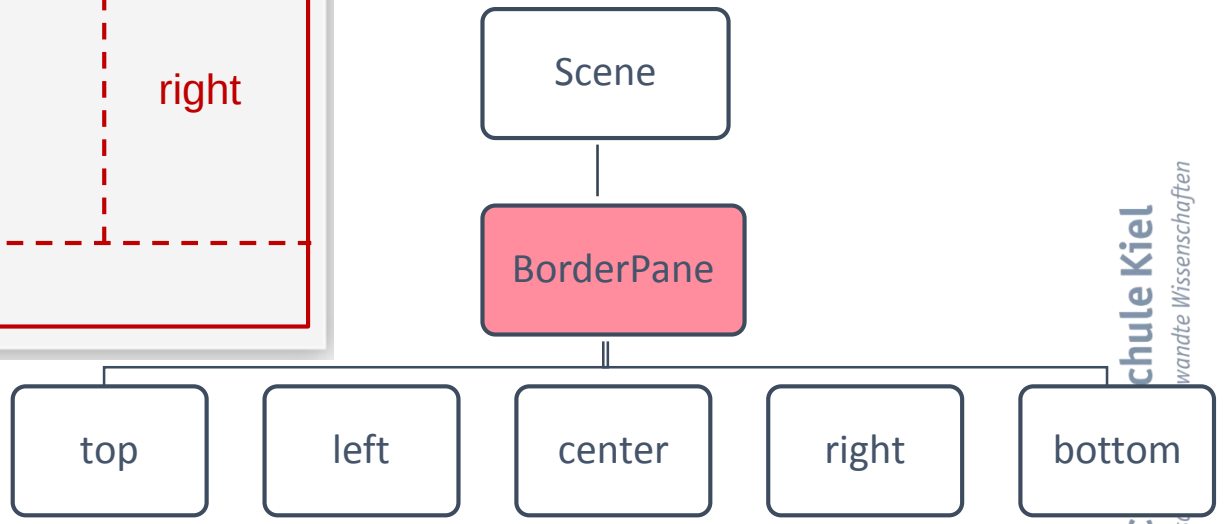
- Statt eines Labels kann wiederum ein neuer Pane etc. hinzugefügt werden => baumartiger hierarchischer Aufbau

Container, die die Anordnung ihrer Komponenten bestimmen

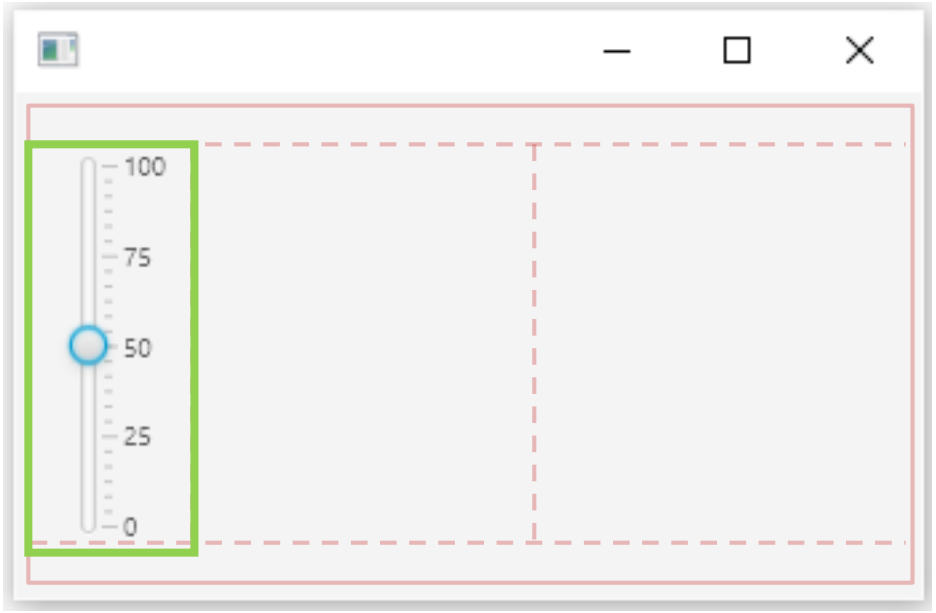


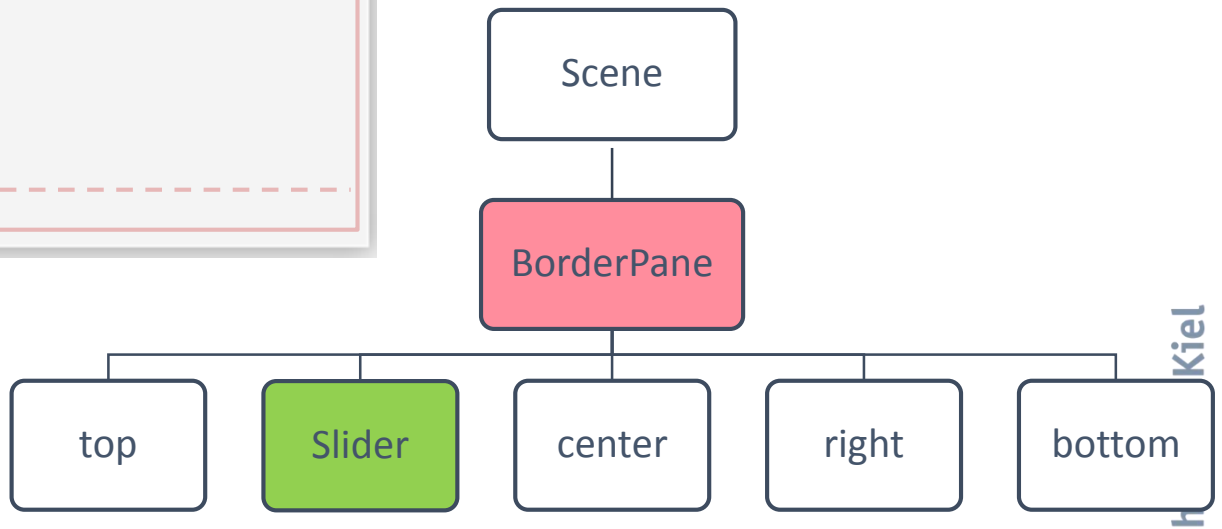
Szene-Graph - Ein Beispiel



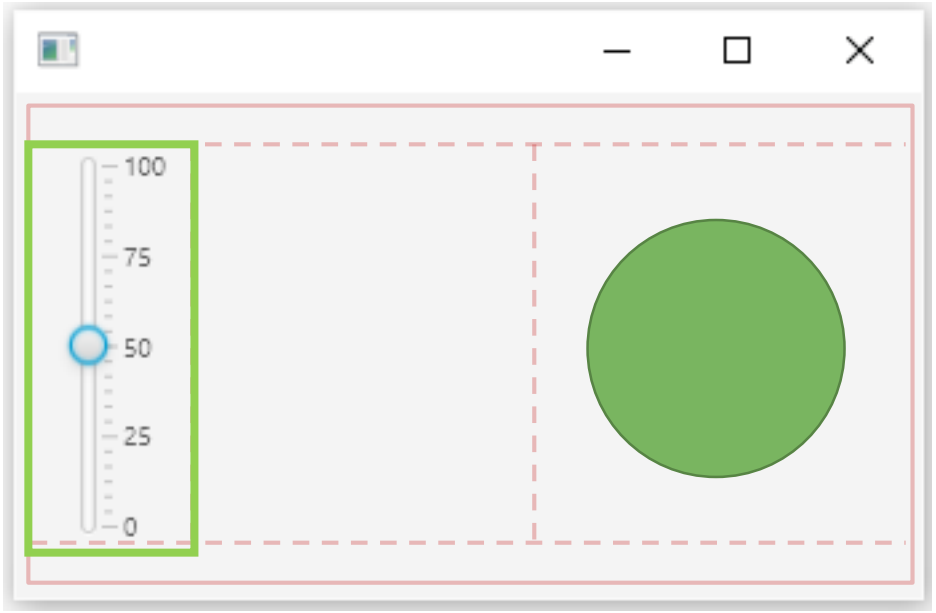


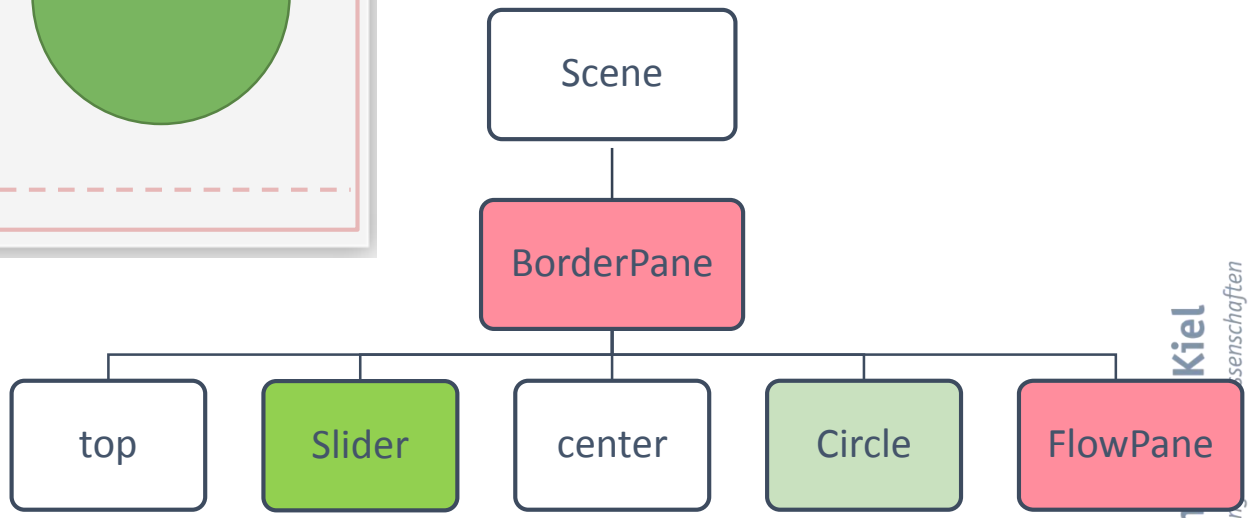
Szene-Graph - Ein Beispiel





Szene-Graph - Ein Beispiel





JavaFX – weitere Eigenschaften

- zusätzlich zu dem Listener-Konzept gibt es noch das Konzept des sogenannten Property-Bindings.
  - Bei dem werden Variablen miteinander verbunden, so dass bei Änderungen an einer der Variablen der Wert der anderen ebenfalls geändert wird.
- Schreiben deklarativer GUIs, die nicht mit Java, sondern mit XML definiert werden -> FXML
  - Mit SceneBuilder fxml „schreiben“
- das Aussehen der GUI-Komponenten über CSS-Regeln anpassen.
  - CSS (Cascading Style Sheets) wird verwendet, um das Aussehen von Webseiten unabhängig von dem HTML-Code zu halten

Literatur

- Abts, Dietmar: Grundkurs JAVA,Wiesbaden Springer Vieweg, 2015, Online verfügbar
- Christian Ullenboom: Java ist auch eine Insel, Galileo Computing, 2016
- R. Steyer, Einführung in JavaFX, 2014
- Java Tutorial: Creating a GUI with Swing, http://docs.oracle.com/javase/tutorial/uiswing/
- Java Tutorial: Creating a JavaFX GUI, https://docs.oracle.com/javafx/index.html
- http://www.java2s.com/Tutorials/Java/Java_Swing/index.htm
- http://www.java2s.com/Tutorials/Java/JavaFX/index.htm