universität innsbruck

Grafische Benutzeroberflächen mit JavaFX

Programmiermethodik

Lukas Kaltenbrunner, Simon Priller
Universität Innsbruck

GUI

- GUI = Graphical User Interface
  - Grafische Benutzeroberfläche
  - Steuerung der Anwendung durch Maus- und Tastatur mit Hilfe von GUI-Komponenten (Fenster, Buttons, Slider, Scrollbar etc.)
- Wichtige Konzepte bei der GUI-Programmierung
  - Komponenten
    - Auswahl adäquater Komponenten
  - Layout
    - Anordnung der Komponenten am Bildschirm
  - Ereignisgesteuerte Programmierung
    - Komponenten haben eigene Funktionalität.
    - Komponenten reagieren auf Ereignisse (Events).
    - Abfolge der Ereignisse (z.B. Benutzereingabe) bestimmt den Programmablauf!

JavaFX

JavaFX

- API für die Erstellung von GUIs
  - Läuft Standalone
  - Multimedia-Inhalte
  - CSS (Cascading Style Sheet)

- Ursprünglich eigenständig entwickelt
- Seit Java 1.7 und JavaFX 2.2 mitgeliefertes Graphics-Toolkit
- Seit Java 1.8 Versionsnummer angepasst (JavaFX 8)
- Ab Java 11 wieder eigenständiges Projekt

JavaFX-Grundlagen

- Jede JavaFX Applikation erbt von `javafx.application.Application`

```
public class HelloWorldApplication extends Application
```

- Application-Klasse stellt Umgebung (Fenster, Systemmenü und Standardschaltflächen) zur Verfügung

- GUI ist immer hierarchisch aufgebaut

JavaFX-Aufbau

Eine GUI ist immer hierarchisch aufgebaut.

- Stage: Fenster und damit Hauptcontainer (Rand, Buttons zum Minimieren, Maximieren, Schließen)
- Scene-Objekt: Inhaltscontainer (kann ausgetauscht werden)
- Pane-Objekte: Untercontainer, der die UI-Elemente (Widgets) enthält

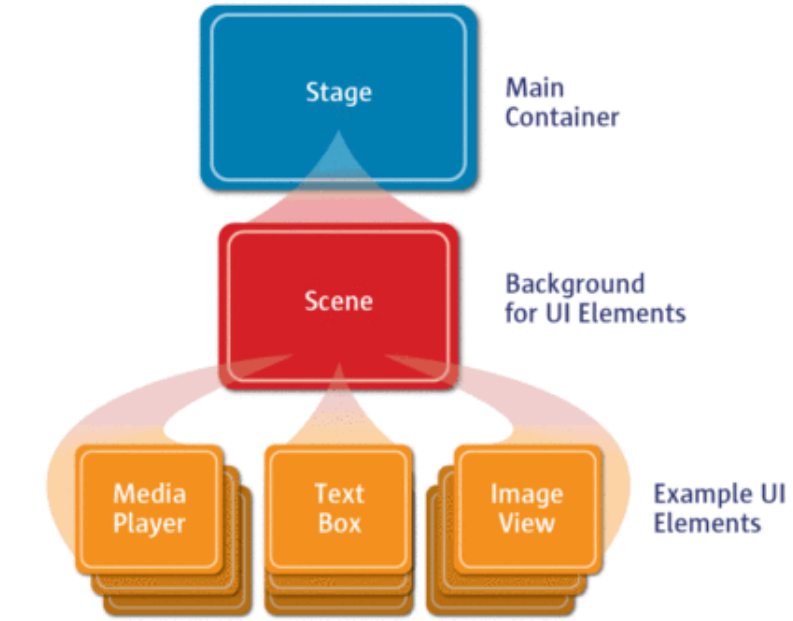
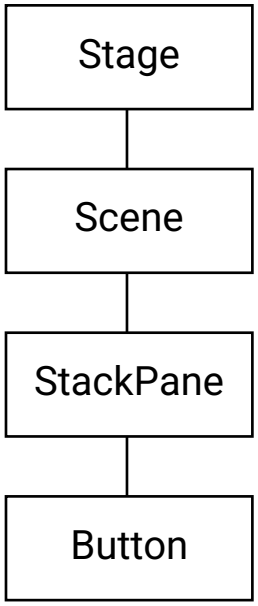
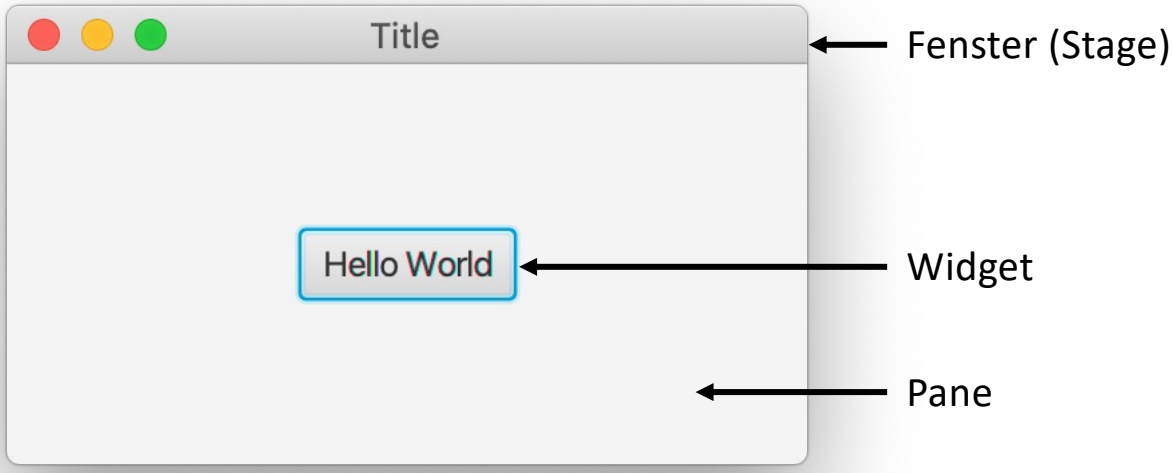


Image Source: http://www.oracle.com

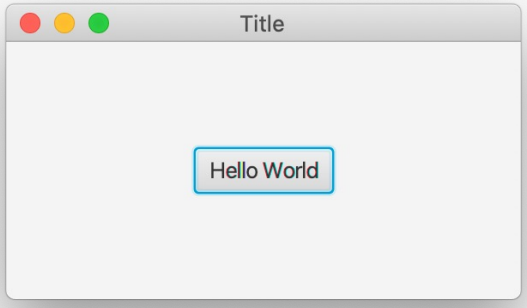
Beispiel: Hello World (1)





Beispiel: Hello World (2)

```
public class HelloWorldApplication extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        Button helloWorld = new Button("Hello World");

        Pane root = new StackPane();
        root.getChildren().add(helloWorld);

        stage.setScene(new Scene(root, 300, 150));
        stage.setTitle("Title"));
        stage.show();
    }

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

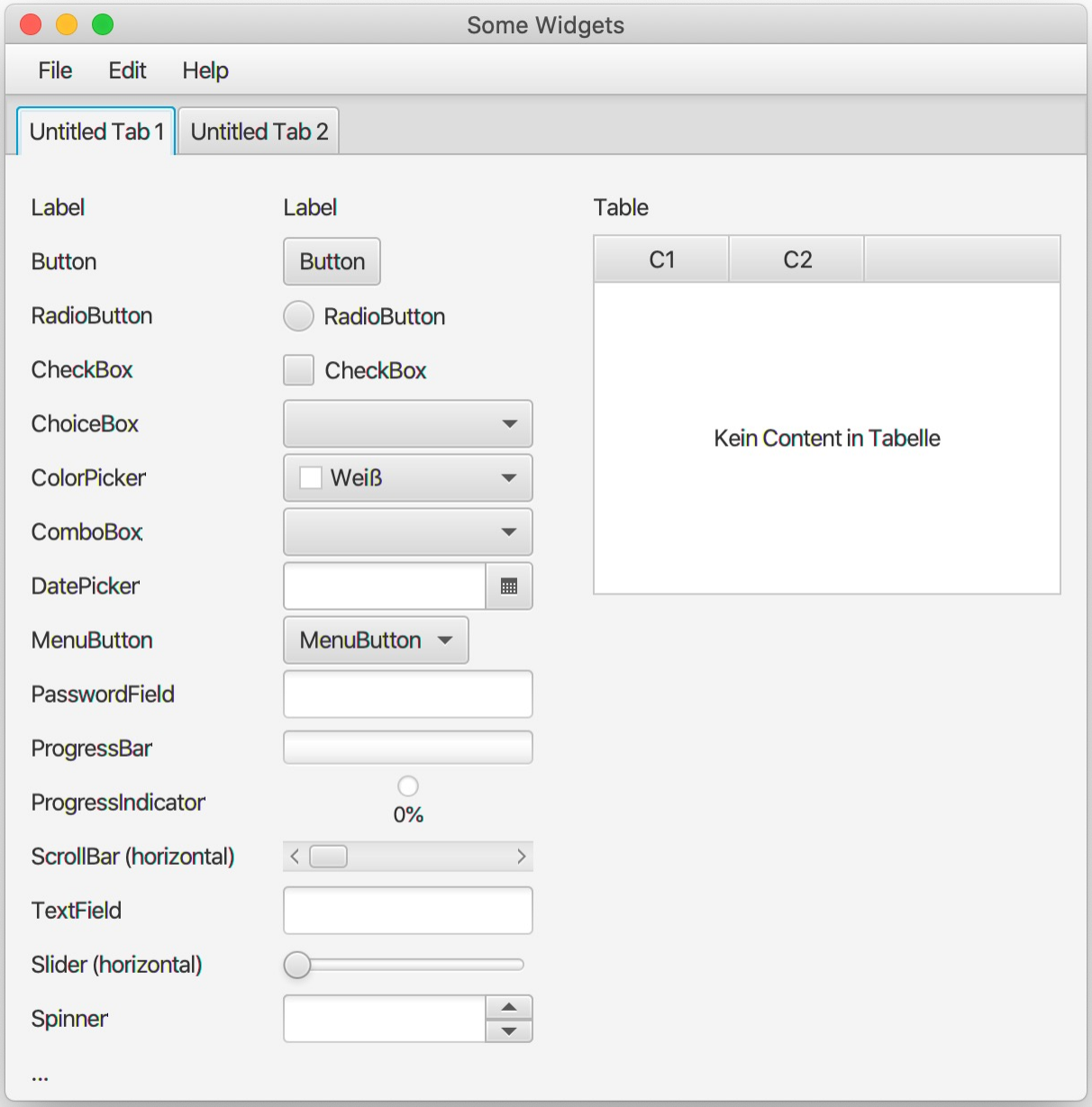


src/at/ac/uibk/pm/gui/basics/HelloWorldApplication.java

Beispiel: Hello World (3)

- `start()`-Methode
  - Die `start`-Methode wird implementiert.
  - `start` wird beim Erzeugen der Anwendung automatisch aufgerufen.
  - In der `start`-Methode wird der Inhalt des Fensters festgelegt.
  - Der Parameter `stage` wird von JavaFX automatisch erzeugt.
- Stage
  - `setTitle()` – setzt Bezeichnung des Fensters in Header.
  - `show()` – zeigt definiertes Fenster an.
- StackPane
  - Container für UI-Elemente (in diesem Fall Button)
  - wird der Scene hinzugefügt
- Scene – Container für Inhalt des Fensters
- `launch` startet die Anwendung

Widgets (1)


Some Widgets
File
Edit
Help
Untitled Tab 1
Untitled Tab 2
Label
Label
Table
Button
Button
C1
C2
RadioButton
RadioButton
CheckBox
CheckBox
ChoiceBox
ColorPicker
Weiß
ComboBox
DatePicker
MenuButton
MenuButton
PasswordField
ProgressBar
ProgressIndicator
0%
ScrollBar (horizontal)
TextField
Slider (horizontal)
Spinner
...
Kein Content in Tabelle

Widgets (2)

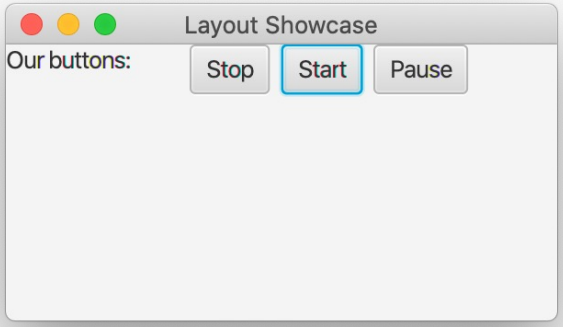
- JavaFX stellt eine Vielzahl von Widgets zur Verfügung.
- Widgets bieten zudem noch Methoden an, mit denen sie Manipuliert werden können. Beispielsweise kann der eingegebene Text abgefragt werden.
- Ablaufsteuerung basiert auch zum Teil auf Widgets (was passiert, wenn Button gedrückt wurde)
  → ereignisbasierte Programmierung

Layouts

Motivation (1)

- Layout bestimmt die Positionierung von Elementen in der GUI
- Naiver Ansatz: absolute Positionierung der Elemente mittels x- und y-Koordinaten

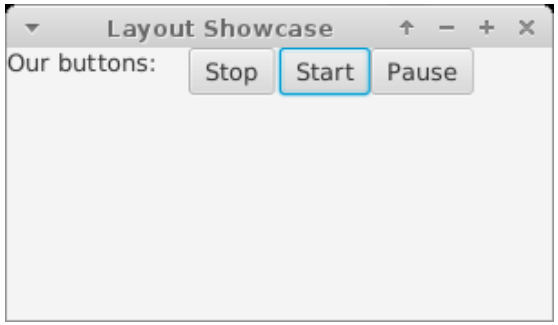
```
…
buttonStop.setLayoutX(100.0);
buttonStop.setLayoutY(0.0);

buttonStart.setLayoutX(150.0);
buttonStart.setLayoutY(0.0);

buttonPause.setLayoutX(200.0);
buttonPause.setLayoutY(0.0);

label.setLayoutX(0);
label.setLayoutY(0);

Pane root = new Pane();
root.getChildren().addAll(label, buttonStart, buttonStop, buttonPause);
…
```



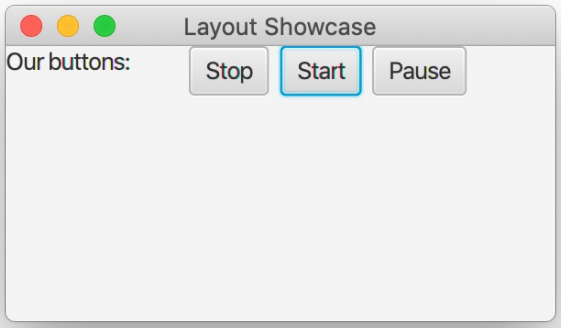
src/at/ac/uibk/pm/gui/basics/layouts/AbsolutePositioningShowcase.java

Motivation (2)

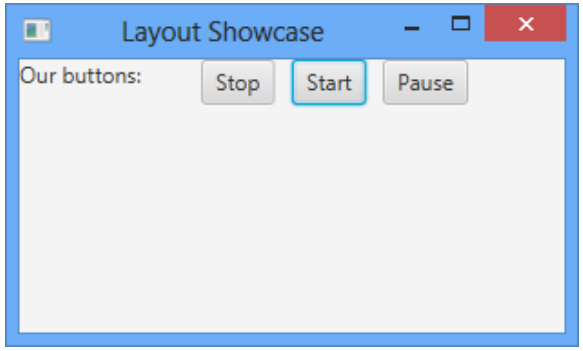
- Probleme:
  - Plattformunabhängigkeit (sieht Applikation in Windows, Linux und MacOS noch gleich aus?)
  - Fedora 28:



  - macOS 10.15:



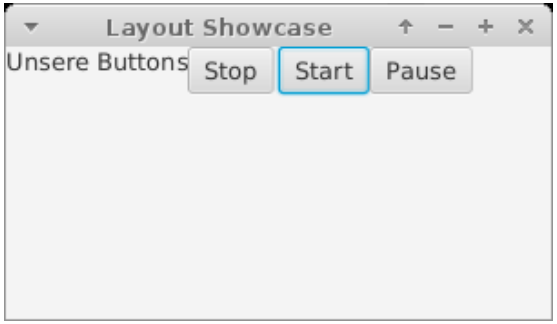
  - Windows 7:



Motivation (3)

- Skalierung („Aufziehen“ des Fensters)

- Internationalisierung (Elemente in verschiedenen Sprachen)



- Lösung:
  - Flexible Layoutmanager von JavaFX verwenden

Layouts

- Flexible Anordnung der UI-Elemente in den Panes
- Verschiedene Panes verfügen über verschiedene Layouts (z.B. FlowPane, BorderPane, etc.)
- Mit Skalierung des Fensters werden auch alle UI-Elemente skaliert und neu positioniert.
- Skalierung und Größe:
  - Initiale Größe der Scene kann im Konstruktor übergeben werden.
  - Größe der Widgets: minimale, maximale und bevorzugte Größe (prefWidth, prefHeight) können angegeben werden.
  - Die angegebenen Größen der Widgets werden vom Layout Manager abgefragt und bei Bedarf verwendet.

BorderPane (1)

- Pane wird in fünf Bereiche unterteilt.
- Pro Bereich ein Widget.
- Die Top- und Bottom-Bereiche werden in ihrer bevorzugten Höhe erzeugt. Die Breite entspricht der gesamten, verfügbaren Weite.
- Left- und Right-Bereiche werden in ihrer bevorzugten Breite erzeugt. Die Höhe entspricht der zwischen Top und Bottom verbleibenden Distanz.
- Bei der Skalierung wird Center entsprechend vergrößert bzw. verkleinert.
- Falls das Fenster zu klein ist, kommt es zu Überlappungen.

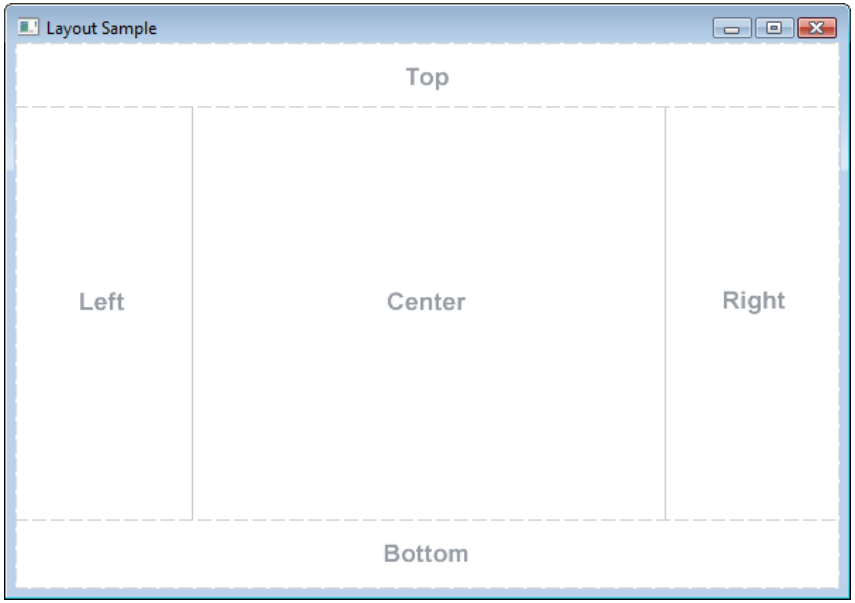
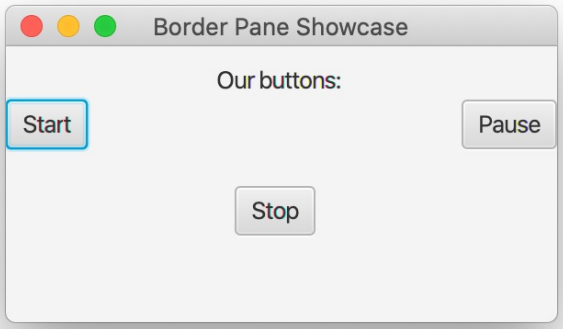


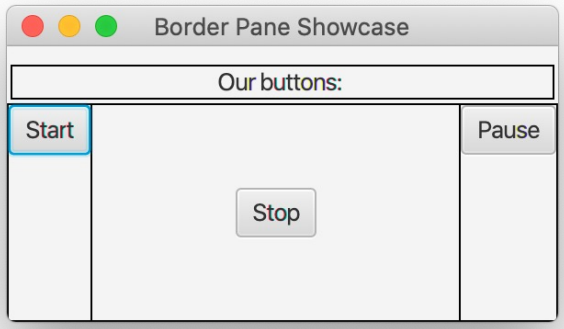
Abbildung übernommen von https://openjfx.io/javadoc/16/javafx.graphics/javafx/scene/layout/BorderPane.html

BorderPane (2)

```
…
BorderPane root = new BorderPane();
root.setTop(label);
BorderPane.setAlignment(label, Pos.CENTER);
BorderPane.setMargin(label, new Insets(10.0, 2.0, 2.0, 2.0));
root.setLeft(buttonStart);
root.setCenter(buttonStop);
root.setRight(buttonPause);
…
```

Insets = „Inside Offsets“ für alle vier Seiten eines Rechteckes (top, right, bottom, left).



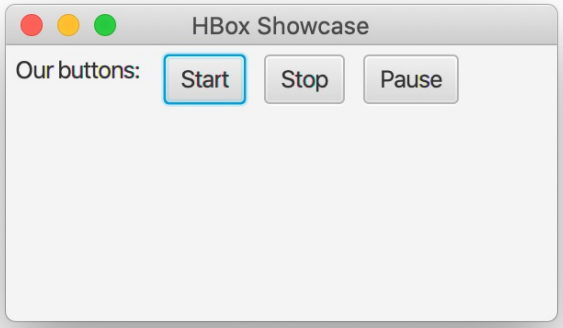


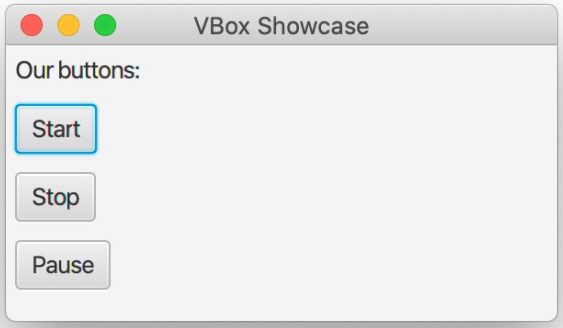
src/at/ac/uibk/pm/gui/basics/layouts/BorderPaneShowcase.java

Hbox, VBox

- Widgets werden in einer Reihe bzw. Spalte angeordnet.
- Reihenfolge der Widgets entspricht der Einfügereihenfolge.
- Bei Skalierung kann angegeben werden, ob und welche Komponenten wachsen bzw. schrumpfen (grow priority).

```
HBox root = new HBox();
root.setSpacing(10.0);
root.setPadding(new Insets(5.0));
root.getChildren().addAll(label, buttonStart, buttonStop, buttonPause);
```





src/at/ac/uibk/pm/gui/basics/layouts/HBoxShowcase.java

src/at/ac/uibk/pm/gui/basics/layouts/VBoxShowcase.java

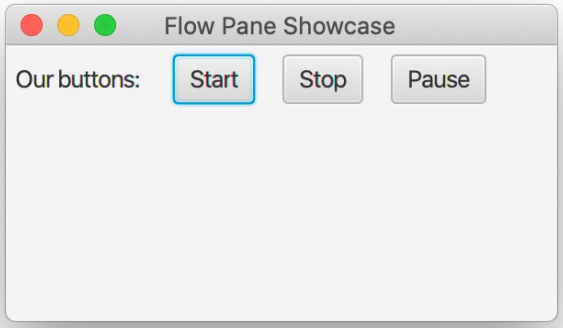
FlowPane (1)

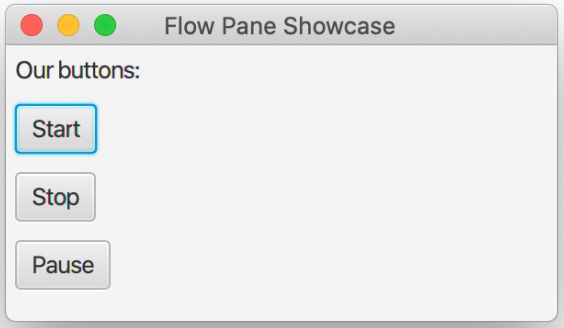
- FlowPane ordnet Widgets nacheinander an (abhängig vom im Pane zur Verfügung stehenden Platz).
- Horizontale FlowPane
  - Widgets horizontal nebeneinander angeordnet (Reihe).
  - Zeilenumbruch beim Überlauf.
- Vertikale FlowPane
  - Widgets vertikal untereinander angeordnet (Spalte).
  - Spaltenumbruch beim Überlauf.

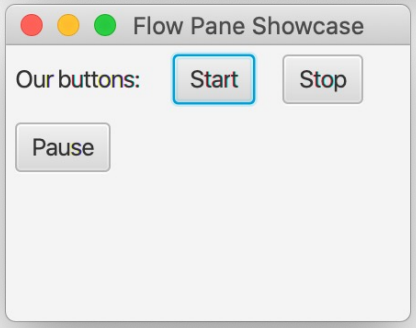
src/at/ac/uibk/pm/gui/basics/layouts/FlowPaneShowcase.java

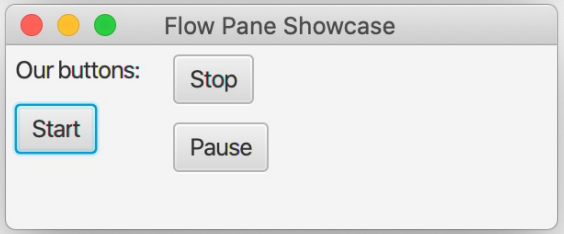
FlowPane (2)

```
FlowPane root = new FlowPane();
// vertical FlowPane
//FlowPane root = new FlowPane(Orientation.VERTICAL);
root.setPadding(new Insets(5.0));
root.setHgap(15.0);
root.setVgap(10.0);
root.getChildren().addAll(label, buttonStart, buttonStop, buttonPause);
```





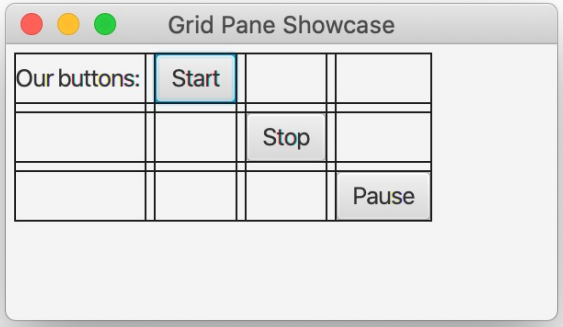




GridPane

- Erzeugt ein Raster aus Zeilen und Spalten
- Widgets können in Zellen eingefügt werden.

```
GridPane root = new GridPane();
root.setPadding(new Insets(5.0));
root.setHgap(5.0);
root.setVgap(5.0);
root.add(label, 0, 0);
root.add(buttonStart, 1, 0);
root.add(buttonStop, 2, 1);
root.add(buttonPause, 3, 2);
root.setGridLinesVisible(true);
```



src/at/ac/uibk/pm/gui/basics/layouts/GridPaneShowcase.java

Weitere Layouts

- StackPane
  - „Stapelt“ Widgets übereinander.
- AnchorPane
  - Widgets können an vier Seiten der Pane „verankert“ werden
  - Bei Skalierung wird die Abstände vom Ankerpunkt beibehalten.
- TilePane
  - Ähnliches Verhalten wie FlowPane
  - Unterteilung in „Tiles“ gleicher Größe (horizontaler oder vertikaler Zeilenumbruch)

Verschachtelung

- Panes können beliebig verschachtelt werden.
- Beispiel: BorderPane mit
  - HBox TOP
    - StackPane
  - VBox LEFT
  - FlowPane RIGHT
  - AnchorPane CENTER
    - GridPane anchored TOP
    - HBox anchored BOTTOM RIGHT

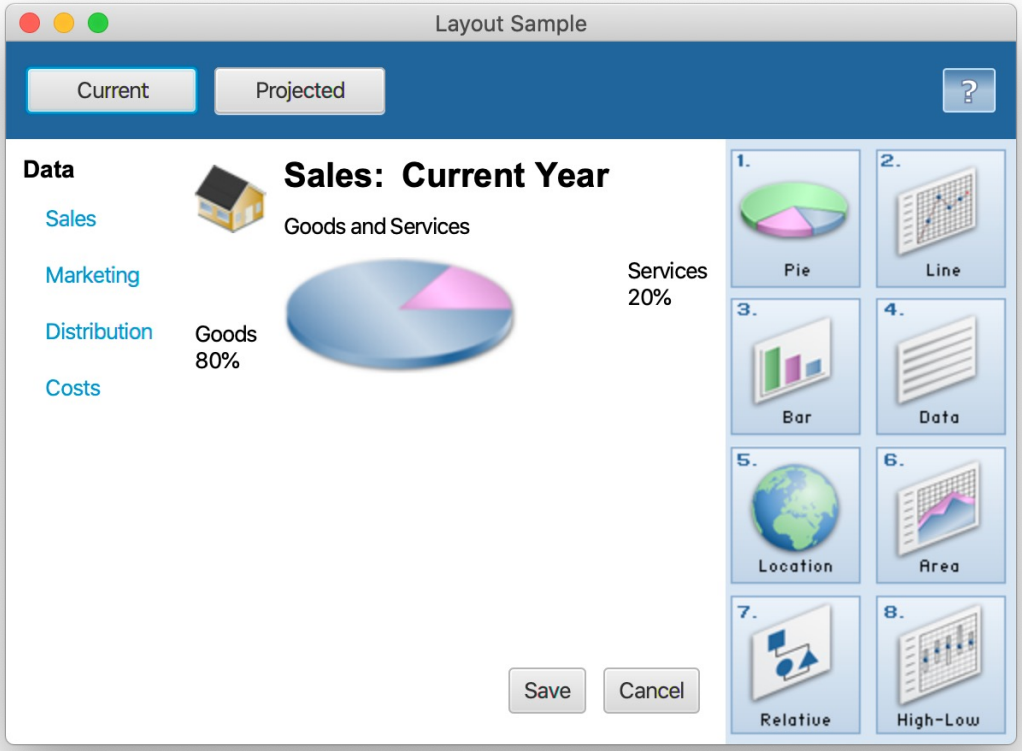


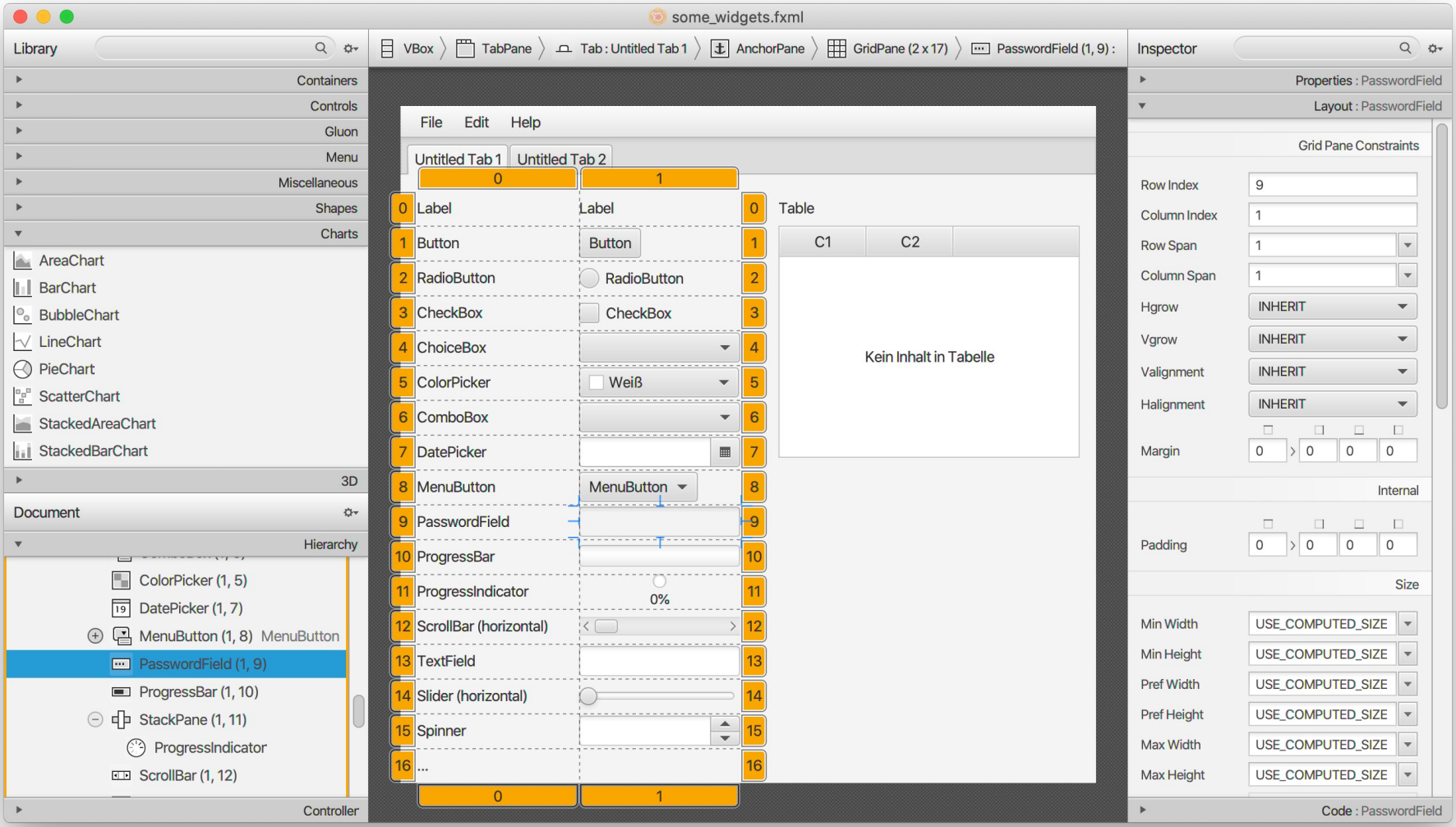
Abbildung übernommen von https://docs.oracle.com/javafx/2/layout/builtin_layouts.htm

SceneBuilder (1)

- Einfaches Designen von GUIs
  - Standalone-Version von Gluon verfügbar

- Erzeugt eine FXML-Datei.
- FXML kann in Java dann mittels der `FXMLLoader`-Klasse geladen werden und bestimmt das Layout der Applikation.

SceneBuilder (2)



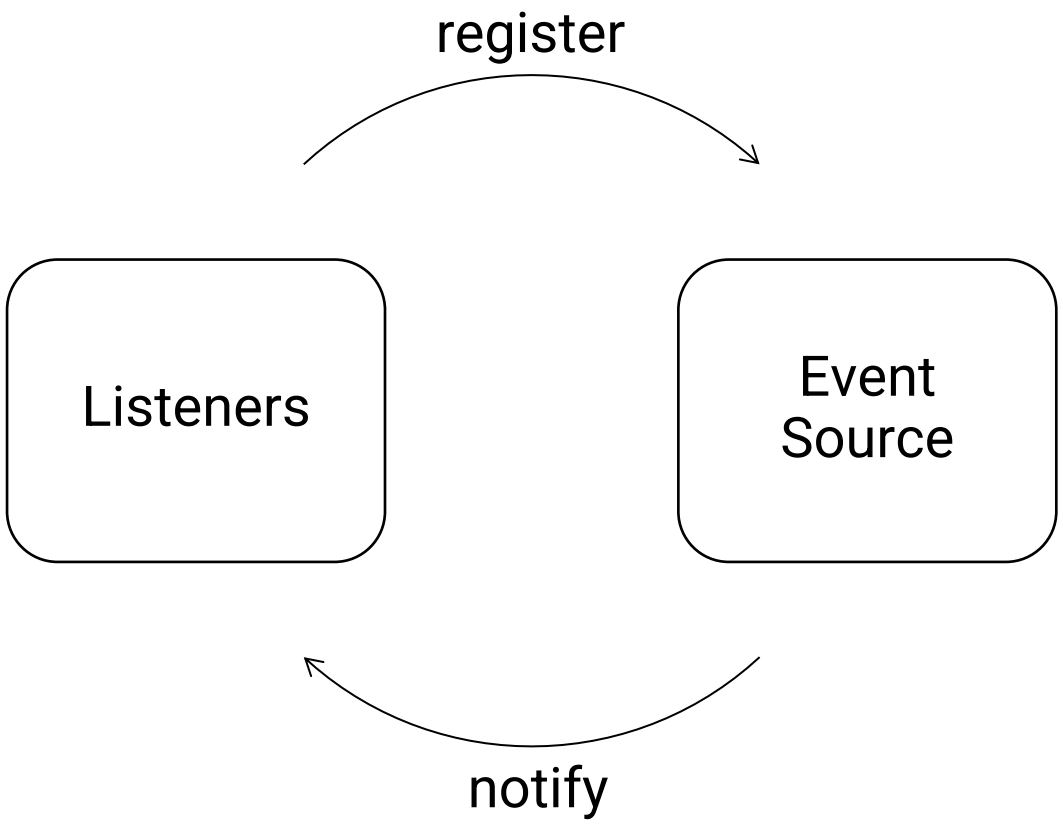
Ereignisbasierte Programmierung

Ereignisbasierte Programmierung

- Der Kontrollfluss eines Systems wird durch Ereignisse und nicht durch Kontrollstrukturen oder Funktionsaufrufe bestimmt.
- Beispiele für Ereignisse
  - Mausklick
  - Eingabe per Tastatur etc.
  - Selektion
  - Drag and Drop
- Typische Anwendung
  - Interaktive Systeme
    - Interaktion zwischen Benutzer und grafischer Benutzerschnittstelle
  - Programmierung von Sensoren
    - Messwerte bestimmen weiteren Ablauf

Generelle Idee

- Zuhörer (Listener) registrieren sich für bestimmte Ereignisse.
- Die Quelle schickt eine Benachrichtigung wenn ein Ereignis eintritt.



Event-Objekte (1)

- Ein Ereignis wird durch ein Objekt repräsentiert.
  - Es gibt unterschiedliche Ereignistypen, von denen Objekte angelegt werden können.
  - z.B. Klick auf Button erzeugt `ActionEvent`
  - Alle Ereignistypen erben von `javafx.event.Event`.
  - Ereignisse werden meist von Widgets erzeugt.

- Ein Ereignisobjekt beinhaltet Informationen über
  - Ereignistyp (Event Type)
  - Source (Quelle des Ereignisses (z.B. auf welchen Button wurde geklickt?))
  - Target Ereignis (Handler)

Event-Objekte (2)

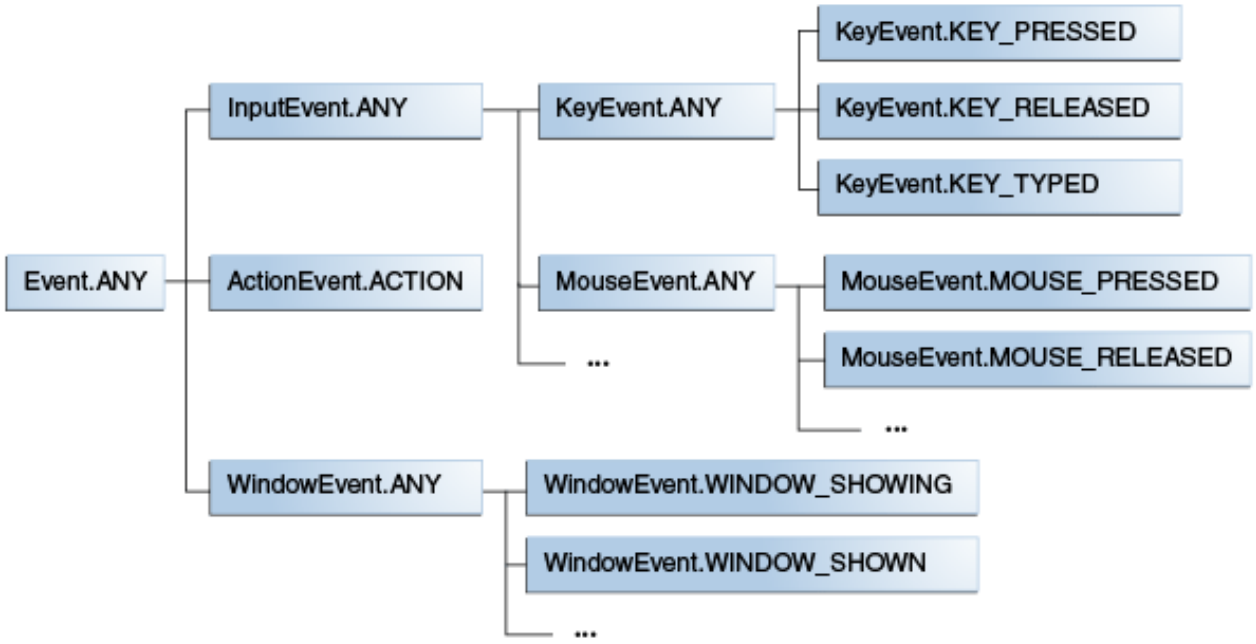


Abbildung übernommen aus https://docs.oracle.com/javafx/2/events/processing.htm

Listener

- Möchte ein Objekt von einer Komponente über ein bestimmtes Ereignis benachrichtigt werden, dann muss die Komponente:
  - Das EventHandler Interface implementieren.
  - Sich als Listener für das Event bei der Komponente (Source) registrieren.
- Functional Interface:

```
public interface EventHandler <T extends Event> extends EventListener {
    void handle(T event);
}
```

- Methode `handle()` reagiert auf auftretendes Ereignis.
- Diese Implementierung wird als **Event Handler** bezeichnet.
  - Meist als als innere (anonyme) Klasse oder Lambda implementiert.

Event Handler

- Handler als Listener registrieren mittels setOn<EventTyp>-Methode des zu überwachenden Widgets
- Beispiel Button wird gedrückt (inner class oder Lambda-Expression):

```
button.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        …
    }
});
```

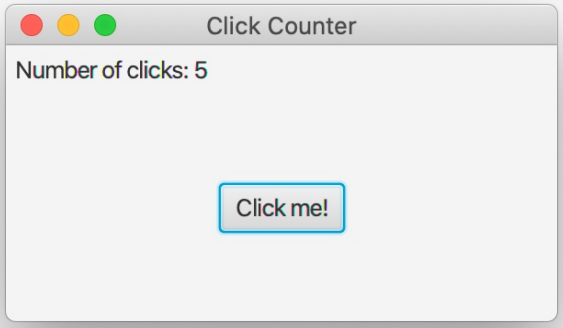
```
button.setOnAction(event -> …);
```

Event-Arten

- ActionEvent
  - Benutzer*in klickt z.B. einen Button an, drückt die Enter-Taste in einem Textfeld oder wählt einen Menüeintrag aus.
- InputEvent
  - Benutzer*in drückt Maustaste, Tastatur-Taste.
- KeyEvent (Subklasse von InputEvent)
  - Benutzer*in klickt eine Maustaste während der Mauszeiger über einer Komponente ist.
- MouseEvent (Subklasse von InputEvent)
  - Benutzer*in bewegt die Maustaste über einer Komponente, drag and drop, Maustasten drücken.
- TouchEvent
  - Benutzer*in bedient Applikation mittels Touch-Eingabe.
- WindowEvent
  - Fenster wird maximiert, minimiert, skaliert.

Beispiel – Mausklick auf Button

```
@Override
public void start(Stage stage) {
    Label label = new Label(getClicksText());
    Button helloWorld = new Button();
    helloWorld.setText("Click me!");

    helloWorld.setOnAction(event -> {
        ++clickCounter;
        label.setText(getClicksText());
    });
    …
}
```



src/at/ac/uibk/pm/gui/events/ClickCountApplication.java

Threads

- Die Ereignisbehandlung erfolgt im main-Thread.
  - Ein Event-Handler nach dem anderen.
- Auch die Zeichenroutinen der Komponenten laufen im main-Thread.
  - Während ein Event Handler abgearbeitet wird, ist die GUI „eingefroren“ (reagiert nicht).
- Event Handler sollten immer nur wenig Rechenzeit verbrauchen!
  - Langfristige Berechnungen müssen asynchron in extra Threads ausgeführt werden.

Weitere Funktionalitäten von JavaFX

- Animationen
- Grafiken zeichnen
- Diagramme zeichnen
- CSS zum Stylen der Anwendung
- Gesten unterstützen für mobile Devices (Tablets, Smartphone)