

# FOP Recap #14



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

## GUIs





---

# Hey!



## ■ GUIs

- ▣ Wie? Was? GUI?
- ▣ Was war nochmal JavaFX?
- ▣ Application, Stage, Scene, Scene Graph
- ▣ LayoutManagers
- ▣ Wichtige Elemente des Scene Graphs
- ▣ Listeners
- ▣ Bindings

## ■ Canvas - GraphicsContext

- ▣ Zeichenmethoden
- ▣ Farbe/Schriftart einstellen



## GUIs

Wie? Was?

Was war nochmal JavaFX?

JavaFX - Application, Stage, Scene, Scene Graph

LayoutManager

JavaFX - Wichtige Elemente des Scene Graphs

Bindings

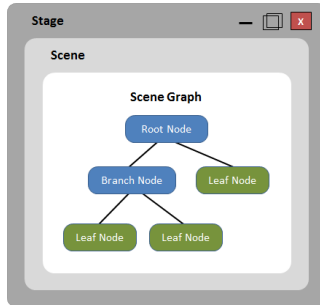
Canvas - GraphicsContext

- GUI steht für Graphical User Interface
- Heißt: Grafische Oberfläche mit Knöpfen und anderem



- JavaFX ist eine Bibliothek für die Entwicklung von GUIs
- Wurde in Java 8 eingeführt
- Wurde in Java 11 als Standard-Bibliothek eingeführt
- Wird in Java 17 als Standard-Bibliothek entfernt

Hauptklasse: erbt von Application





- Legen Position und Größe der verschiedenen Komponenten fest
- Manche erfordern Extra-Parameter beim Verwenden von add
- Häufig genutzte Klassen sind hierbei:
  - ▣ BorderLayout
  - ▣ GridLayout
  - ▣ ...

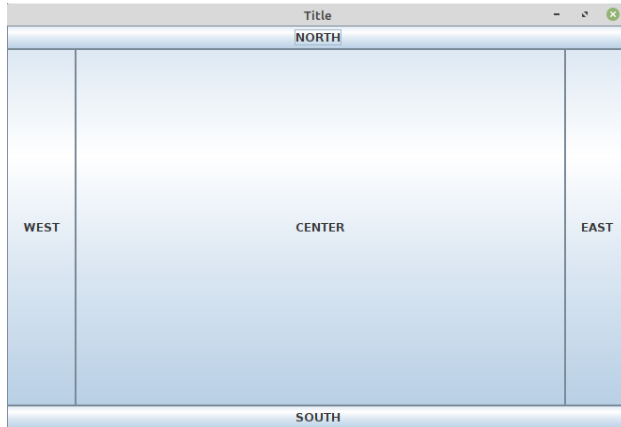




```
1  BorderLayout root = new BorderLayout();  
2  
3  root.setCenter(new Button("CENTER"));  
4  root.setTop(new Button("NORTH"));  
5  root.setBottom(new Button("SOUTH"));  
6  root.setRight(new Button("EAST"));  
7  root.setLeft(new Button("WEST"));
```

# GUIs

## LayoutManager — BorderLayout





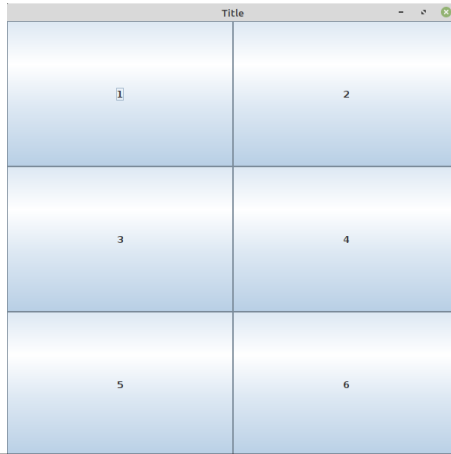
```
1 GridPane root = new GridPane();
2
3 root.add(new Button("1"), 0, 0);
4 root.add(new Button("2"), 1, 0); // column = 1, row = 0
5 root.add(new Button("3"), 2, 0);
6 root.add(new Button("4"), 0, 1);
7 root.add(new Button("5"), 1, 1);
8 root.add(new Button("6"), 2, 1);
```

# GUIs

## LayoutManager — GridPane



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT





Button beschreibt einen Knopf

- Können Aktion bei Klick ausführen (`.setOnAction`)
- Können Text oder Bild anzeigen



Label beschreibt ein Text-Anzeige-Element

- Kann Text anzeigen
- Kann Textformatierung (z.B. fett) haben
- Kann Textfarbe haben



`TextField` beschreibt ein Text-Eingabe-Element

- Kann Text anzeigen
- Kann Textformatierung (z.B. fett) haben
- Kann Textfarbe haben
- Kann Text ändern
- Text kann über `getText()` ausgelesen werden



Slider beschreibt ein Schieberegler-Element

- Kann Wert anzeigen
- Kann Wert ändern
- Wertebereich kann festgelegt werden
- Kann Wert über `getValue()` ausgelesen werden





CheckBox beschreibt ein Checkbox-Element

- Kann für Ja/Nein-Fragen verwendet werden
- Visualisiert Zustand mit Häkchen
- Zustand kann über `isSelected()` ausgelesen werden



VBox und HBox beschreiben Horizontale und Vertikale Container oder Gruppen

- Elemente per `getChildren().add()` hinzufügen
- Zentrieren per `setAlignment()`



```
1 public class MainApp extends Application {
2     public static void main(String[] args) {
3         launch(args);
4     }
5     @Override
6     public void start(Stage primaryStage) throws Exception {
7         // Set title
8         primaryStage.setTitle("First JavaFX Application");
9         // Set minimum size
10        primaryStage.setMinWidth(600);
11        primaryStage.setMinHeight(600);
12        // Set LayoutManager
13        BorderPane root = new BorderPane();
14        // Add Components
15        root.setCenter(new Button("Test"));
16        // Show frame
17        primaryStage.setScene(new Scene(root));
18        primaryStage.show();
19    }
20 }
```

# GUIs

## JavaFX - Wichtige Elemente des Scene Graphs — JavaFX - Beispiel



**Abbildung:** Minimal-Beispiel für JavaFX (je nach Plattform würde oben noch eine Titelleiste erscheinen)



- Werden genutzt um bestimmte Benutzeraktionen zu empfangen
- Häufig genutzte Klassen sind hierbei:
  - ▣ `MouseListener` (für Mauseaktionen)
  - ▣ `KeyListener` (für Tastatureingaben)
  - ▣ `ActionListener` (z.B. für Buttons)
  - ▣ ...



```
1 // ...
2 BorderPane root = new BorderPane();
3
4 Button button = new Button("I was clicked 0 times");
5 int buttonClicks = 0;
6 button.setOnAction(
7     (event) -> {
8         button.setText("I was clicked " + ++buttonClicks + " times");
9     }
10 );
11 root.setCenter(button);
```



Bindings sind eine Möglichkeit, die Daten zwischen verschiedenen Objekten zu synchronisieren. Also konkret: Wenn sich der Wert eines Objektes *a* ändert, soll sich der Wert eines anderen Objektes *b* ebenfalls ändern, dann sagt man, dass *b* an *a* gebunden ist.

- Bindings funktionieren auf Properties
- Entweder unidirektional (`.bind()` *a* -> *b*) oder bidirektional (`.bindBidirectional()` *a* <-> *b*)
- Viele Util-Methoden, siehe Vorlesungsfolien



- Guckt euch den Code gut an
- Guckt euch das JavaDoc der benötigten Klassen an. (Sind teilweise auch auf dem Übungsblatt verlinkt, alternativ in IntelliJ direkt)



# Das steht heute auf dem Plan



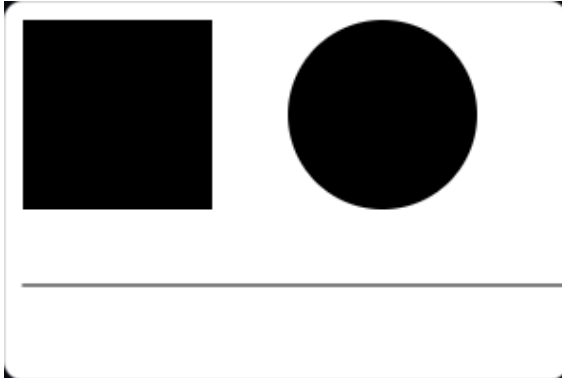
TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

GUIs

Canvas - GraphicsContext

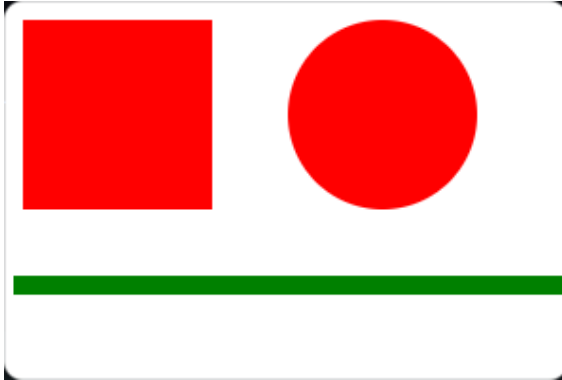


```
1 final Canvas canvas = new Canvas(300, 200);
2 final var gc = canvas.getGraphicsContext2D();
3
4 // draw a rectangle, a circle and a line
5 gc.fillRect(10, 10, 100, 100);
6 gc.fillOval(150, 10, 100, 100);
7 gc.strokeLine(10, 150, 300, 150);
```





```
1 final Canvas canvas = new Canvas(300, 200);
2 final var gc = canvas.getGraphicsContext2D();
3
4 // draw a red rectangle, a circle and a fat, green line
5 gc.setFill(Color.RED);
6 gc.fillRect(10, 10, 100, 100);
7 gc.fillOval(150, 10, 100, 100);
8 gc.setStroke(Color.GREEN);
9 gc.setLineWidth(10);
10 gc.strokeLine(10, 150, 300, 150);
11 gc.setFill(Color.BLACK);
12 gc.setStroke(Color.BLACK);
```





---

# Live-Coding!