



e-Lite



Tecniche di Programmazione – A.A. 2016/2017

# Summary

---

1. About and History
2. Basic concepts
3. Minimal JavaFX Application
4. Application structure
5. The Scene Graph
6. Events



# GUI in Java

---

- ▶ **Graphic framework available in Java**
  - ▶ AWT (1996)
  - ▶ Swing (1998)
  - ▶ Extremely powerful, many extensions available
  - ▶ Complex to master, requires low-level handling
  - ▶ Hard to create visually pleasing applications
- ▶ **Alternatives available**
  - ▶ Most notable: SWT (Eclipse)
  - ▶ Still cumbersome to master
- ▶ **On a different Universe, web-based user interfaces became nicer and faster to create**

# JavaFX 1.0 – forget it

---

- ▶ JavaFX 1 (2008)
- ▶ JavaFX 1 and JavaFX 2 are completely different
- ▶ Version 1 relied on a “scripting language” to describe scenes, with ‘hooks’ to activate Java code
- ▶ JavaFX 1.x is now deprecated

# JavaFX 8 (and JavaFX 2.x)

---

- ▶ Redesigned from scratch
- ▶ The JavaFX 2.x/8.0 framework is entirely written in Java
- ▶ For visual layout, an XML file may also be used (called FXML)
- ▶ Graphic appearance borrows from web-standard CSS style sheets
- ▶ UI programming is based on easy to handle events and bindings
- ▶ Oracle plans to deprecate Swing in favor of JavaFX 2
- ▶ Now called JavaFX 8 (after Java 8 – JDK 1.8)

# Getting and running JavaFX

---

- ▶ JavaFX is already included in Oracle JDK 7 and JDK8
  - ▶ Not in JDK 6.x
  - ▶ Not in OpenJDK (beware, Linux users!)
- ▶ JDK 8 includes significant JavaFX improvements.
- ▶ Recommended:
  - ▶ JavaFX Scene Builder (latest version: 8.1)
  - ▶ Eclipse: e(fx)clipse plugin, available in the Eclipse Marketplace
- ▶ Download links are in the course webpage



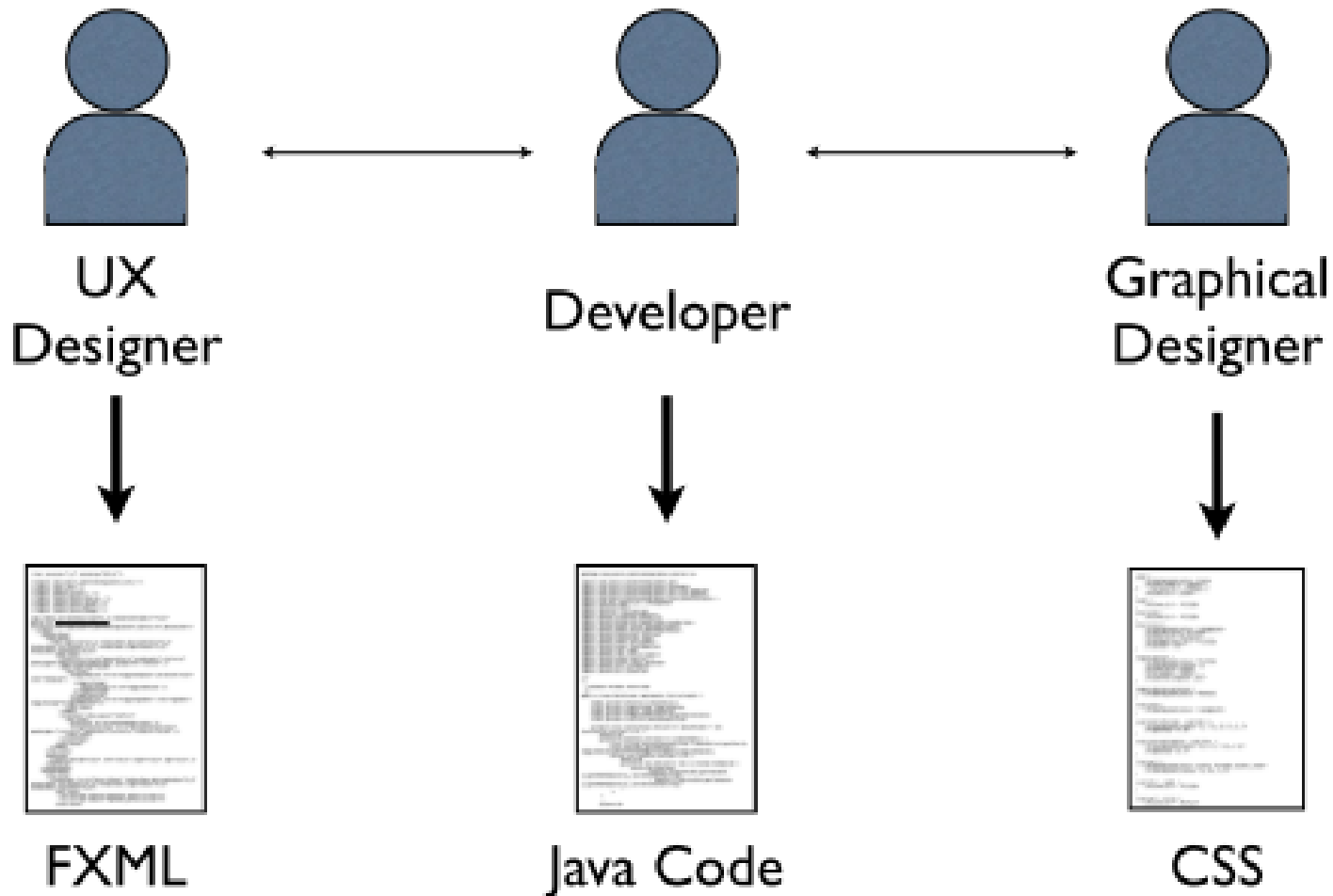
# Basic concepts

## Introduction to JavaFX



# Separation of concerns

---



# Empty JavaFX window

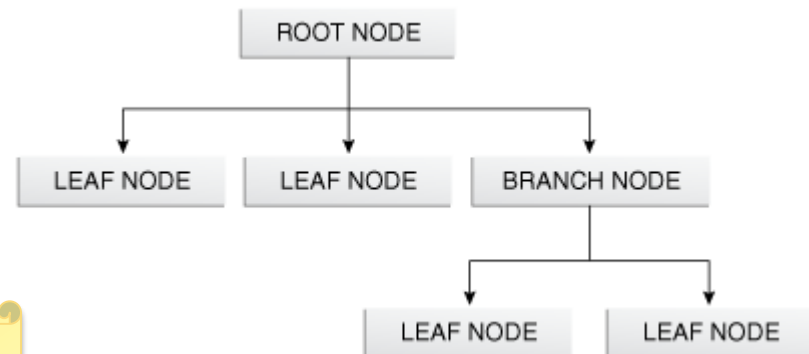
---

```
public class Main extends Application {  
  
    @Override  
    public void start(Stage stage) {  
        Group root = new Group(); // the root is Group or Pane  
        Scene scene = new Scene(root, 500, 500, Color.BLACK);  
        stage.setTitle("JavaFX Demo");  
        stage.setScene(scene);  
        stage.show();  
    }  
  
    public static void main(String[] args) {  
        launch(args);  
    }  
}
```

# Key concepts in JavaFX

---

- ▶ **Stage:** where the application will be displayed (e.g., a Windows' window)
- ▶ **Scene:** one container of Nodes that compose one “page” of your application
- ▶ **Node:** an element in the Scene, with a visual appearance and an interactive behavior. Nodes may be hierarchically nested



My best friend is the JavaFX JavaDoc API

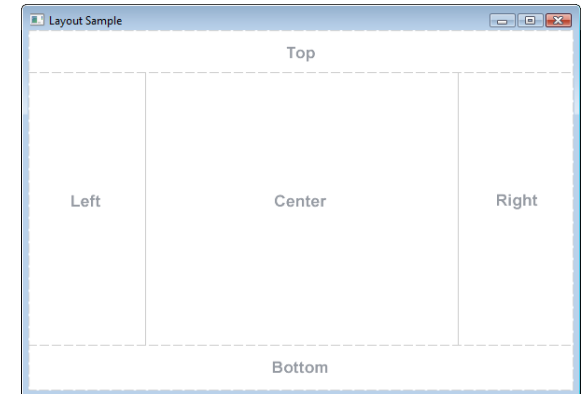
<http://docs.oracle.com/javase/8/javafx/api/>

# Some 'Leaf' Nodes (Controls)



# Some 'Parent' Nodes (Container 'Panes')

- ▶ **BorderPane** (5-areas)
- ▶ **Hbox, VBox** (linear sequence)
- ▶ **StackPane** (overlay all children)
- ▶ **GridPane** (row x columns)
- ▶ **FlowPane** (flowing boxes, wrap around)
- ▶ **TilePane** (flowpane with equally sized boxes)
- ▶ **AnchorPane** (magnetically attach nodes at corners or sides)



# Some Nodes (Charts)



# And more coming...



# How to add scene content

---

## ▶ In Java code

- ▶ By creating and adding new Node subclasses
  - ▶ Standard way, in Java (boring and error-prone)
- ▶ By using node Builder classes
  - ▶ Programming pattern, later on...

## ▶ In FXML

- ▶ By writing XML directly
- ▶ By using the Scene Builder
- ▶ And loading the FXML into the application



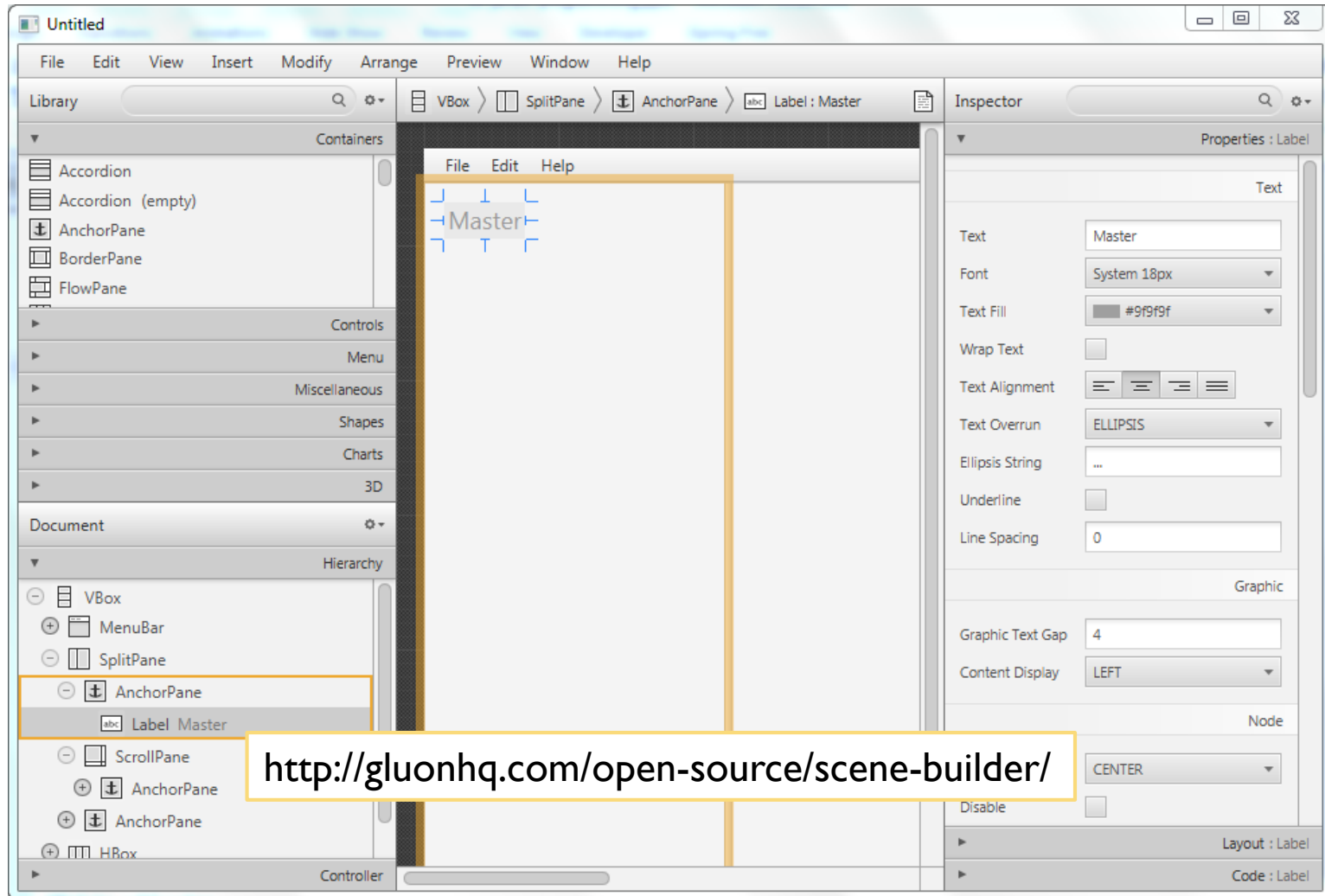
# Adding some shape

---

```
public class Main extends Application {  
  
    @Override  
    public void start(Stage stage) {  
        Group root = new Group();  
  
        Rectangle rect = new Rectangle(25,25,250,250);  
        r.setFill(Color.BLUE);  
        root.getChildren().add(rect);  
  
        Scene scene = new Scene(root, 500, 500, Color.BLACK);  
  
        stage.setTitle("JavaFX Demo");  
        stage.setScene(scene);  
        stage.show();  
    }  
}
```



# JavaFX Scene Builder 8.1



<http://gluonhq.com/open-source/scene-builder/>

# FXML fragment

---

. . .

```
<HBox id="HBox" alignment="CENTER" spacing="15.0"
AnchorPane.rightAnchor="23.0" AnchorPane.topAnchor="22.0">
  <children>
    <Button id="button1" fx:id="newIssue" onAction="#newIssueFired"
      text="New" />
    <Button id="button2" fx:id="saveIssue" onAction="#saveIssueFired"
      text="Save" />
    <Button id="button3" fx:id="deleteIssue" onAction="#deleteIssueFired"
      text="Delete" />
  </children>
</HBox>
<ImageView id="IssueTrackingLite" layoutX="14.0" layoutY="20.0">
  <image>
    <Image url="@IssueTrackingLite.png" preserveRatio="true" smooth="true" />
  </image>
</ImageView>
```

. . .



# Building a scene from FXML

---

```
public void start(Stage stage) throws Exception {  
    Parent root = FXMLLoader.Load(  
        getClass().getResource("circle.fxml"));  
  
    stage.setTitle("Circle Demo");  
    stage.setScene(new Scene(root, 500, 150));  
    stage.show();  
}
```



# Empty JavaFX window

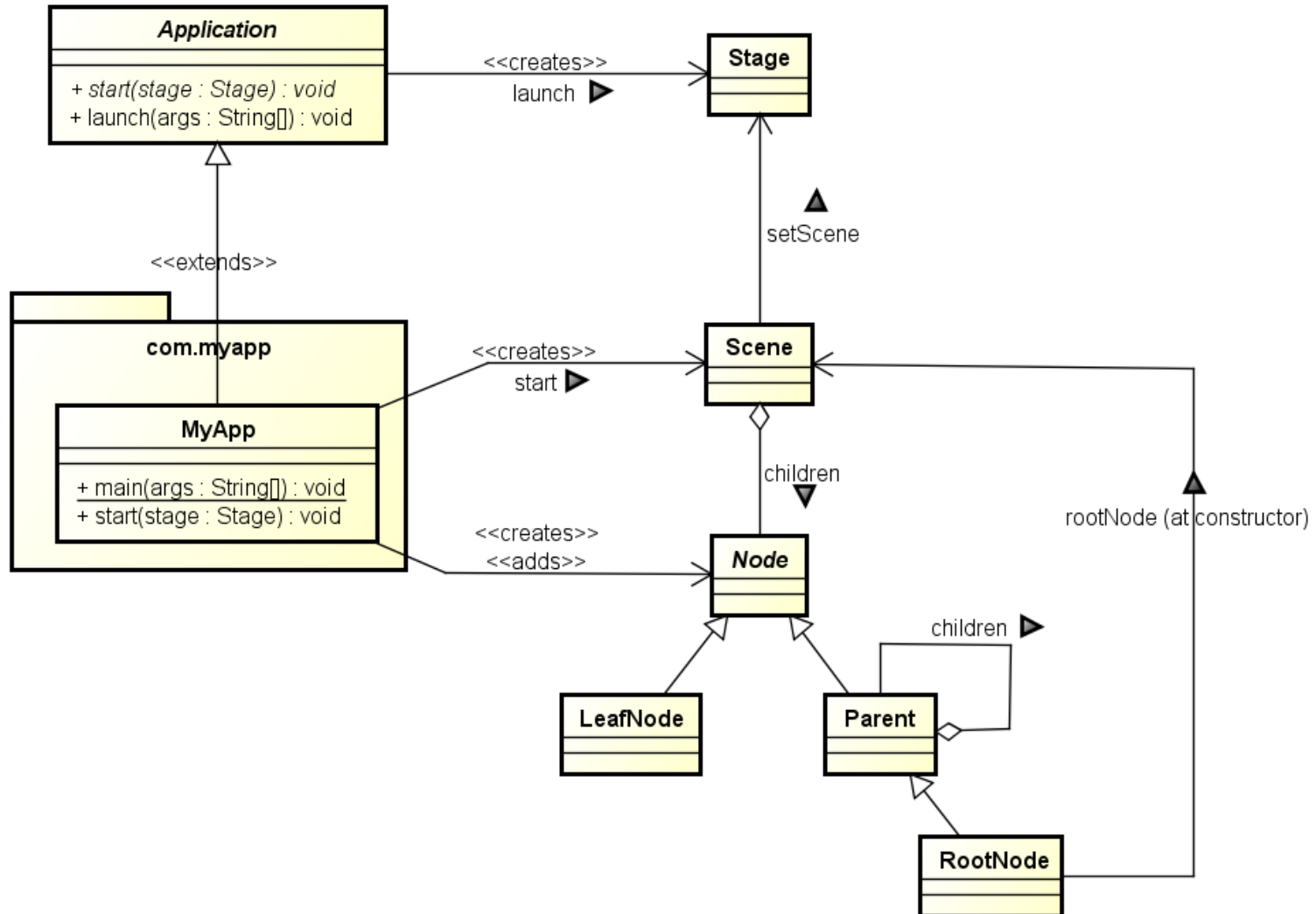
---

```
public class Main extends Application {

    @Override
    public void start(Stage stage) {
        Group root = new Group(); // the root is Group or Pane
        Scene scene = new Scene(root, 500, 500, Color.BLACK);
        stage.setTitle("JavaFX Demo");
        stage.setScene(scene);
        stage.show();
    }

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

# Typical Class Diagram



# General rules

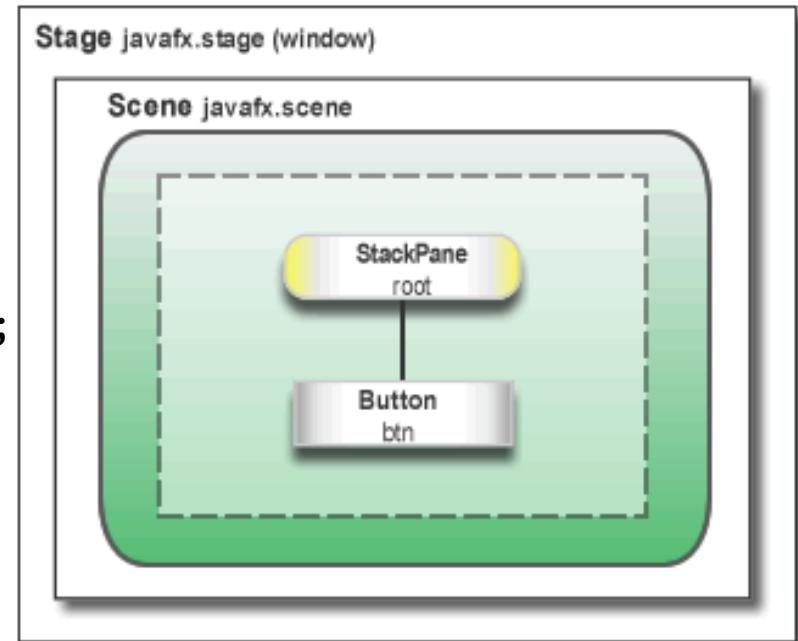
---

- ▶ A JavaFX application extends `javafx.application.Application`
- ▶ The `main()` method should call `Application.launch()`
- ▶ The `start()` method is the main entry point for all JavaFX applications
  - ▶ Called with a Stage connected to the Operating System's window
- ▶ The content of the scene is represented as a hierarchical scene graph of nodes
  - ▶ Stage is the top-level JavaFX container
  - ▶ Scene is the container for all content



# Minimal example

```
public class HelloWorld extends Application {  
    public static void main(String[] args) {  
        launch(args);  
    }  
  
    @Override  
    public void start(Stage primaryStage) {  
        primaryStage.setTitle("Hello World!");  
  
        StackPane root = new StackPane();  
  
        Button btn = new Button();  
        btn.setText("Say 'Hello World'");  
  
        root.getChildren().add(btn);  
  
        primaryStage.setScene(new Scene(root, 300, 250));  
        primaryStage.show();  
    }  
}
```



# Stage vs. Scene

---

## `javafx.stage.Stage`

- ▶ The JavaFX Stage class is the top level JavaFX container.
- ▶ The primary Stage is constructed by the platform.
- ▶ Additional Stage objects may be constructed by the application.
- ▶ A stage can optionally have an owner Window.

## `javafx.scene.Scene`

- ▶ The container for all content in a scene graph
- ▶ The application must specify the root Node for the scene graph
- ▶ Root may be Group (clips), Region, Control (resizes)
- ▶ If no initial size is specified, it will automatically compute it

# Nodes

---

- ▶ The Scene is populated with a tree of Nodes
  - ▶ Layout components
  - ▶ UI Controls
  - ▶ Charts
  - ▶ Shapes
- ▶ Nodes have Properties
  - ▶ Visual (size, position, z-order, color, ...)
  - ▶ Contents (text, value, data sets, ...)
  - ▶ Programming (event handlers, controller)
- ▶ Nodes generate Events
  - ▶ UI events
- ▶ Nodes can be styled with CSS

# Events

---

- ▶ FX Event (`javafx.event.Event`):
  - ▶ Event Source => a Node
  - ▶ Event Target
  - ▶ Event Type
- ▶ Usually generated after some user action
- ▶ `ActionEvent`, `TreeModificationEvent`, `InputEvent`, `ListView.EditEvent`, `MediaErrorEvent`, `TableColumn.CellEditEvent`, `TreItem.TreeModificationEvent`, `TreeView.EditEvent`, `WebEvent`, `WindowEvent`, `WorkerStateEvent`
- ▶ You can define **event handlers** in your application

# Properties

- ▶ Extension of the Java Beans convention
  - ▶ May be used also outside JavaFX
- ▶ Encapsulate properties of an object
  - ▶ Different types (string, number, object, collection, ...)
  - ▶ Set/Get
  - ▶ Observe changes
  - ▶ Supports lazy evaluation
- ▶ Each Node has a large set of Properties

Properties	
Type	Property and Description
BooleanProperty	cancelButton A Cancel Button is the button that receives a keyboard VK_ESC press, if no other node in the scene co
BooleanProperty	defaultButton A default Button is the button that receives a keyboard VK_ENTER press, if no other node in the scene
Properties inherited from class javafx.scene.control.ButtonBase	
armed, onAction	
Properties inherited from class javafx.scene.control.Labeled	
alignment, contentDisplay, ellipsisString, font, graphic, graphicTextGap, labelPadding, mnemonicParsing, tex textFill, textOverrun, text, underline, wrapText	
Properties inherited from class javafx.scene.control.Control	
contextMenu, height, maxHeight, maxWidth, minHeight, minWidth, prefHeight, prefWidth, skinClassName, skin, t	
Properties inherited from class javafx.scene.Parent	
needsLayout	
Properties inherited from class javafx.scene.Node	
blendMode, boundsInLocal, boundsInParent, cacheHint, cache, clip, cursor, depthTest, disabled, disable, effe eventDispatcher, focused, focusTraversable, hover, id, inputMethodRequests, layoutBounds, layoutX, layoutY, localToParentTransform, localToSceneTransform, managed, mouseTransparent, onContextMenuRequested, onDragDete onDragDone, onDragDropped, onDragEntered, onDragExited, onDragOver, onInputMethodTextChanged, onKeyPressed, onKeyTyped, onMouseClicked, onMouseDragEntered, onMouseDragExited, onMouseDragged, onMouseDragOver, onMouseD onMouseEntered, onMouseExited, onMouseMoved, onMousePressed, onMouseReleased, onRotate, onRotationFinished, onRotationStarted, onScrollFinished, onScroll, onScrollStarted, onSwipeDown, onSwipeLeft, onSwipeRight, onSw onTouchMoved, onTouchPressed, onTouchReleased, onTouchStationary, onZoomFinished, onZoom, onZoomStarted, opa pickOnBounds, pressed, rotate, rotationAxis, scaleX, scaleY, scaleZ, scene, style, translateX, translateY, t visible	

# Bindings

---

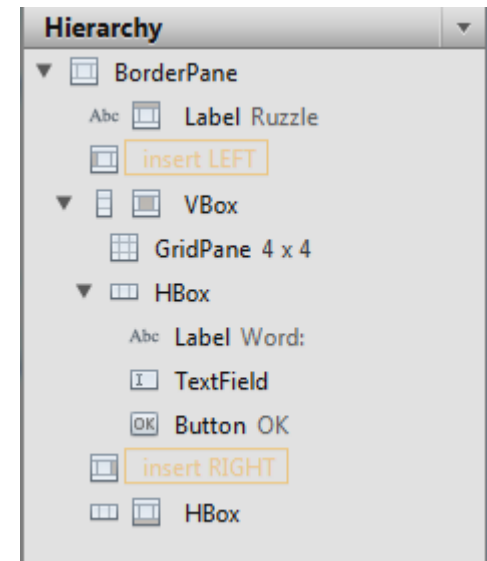
- ▶ Automatically connect («bind») one Property to another Property
  - ▶ Whenever the source property changes, the bound one is automatically updated
  - ▶ Multiple bindings are supported
  - ▶ Lazy evaluation is supported
  - ▶ Bindings may also involve computations (arithmetic operators, if-then-else, string concatenation, ...) that are automatically evaluated
- ▶ May be used to automate UI
- ▶ May be used to connect the Model with the View



# Nodes

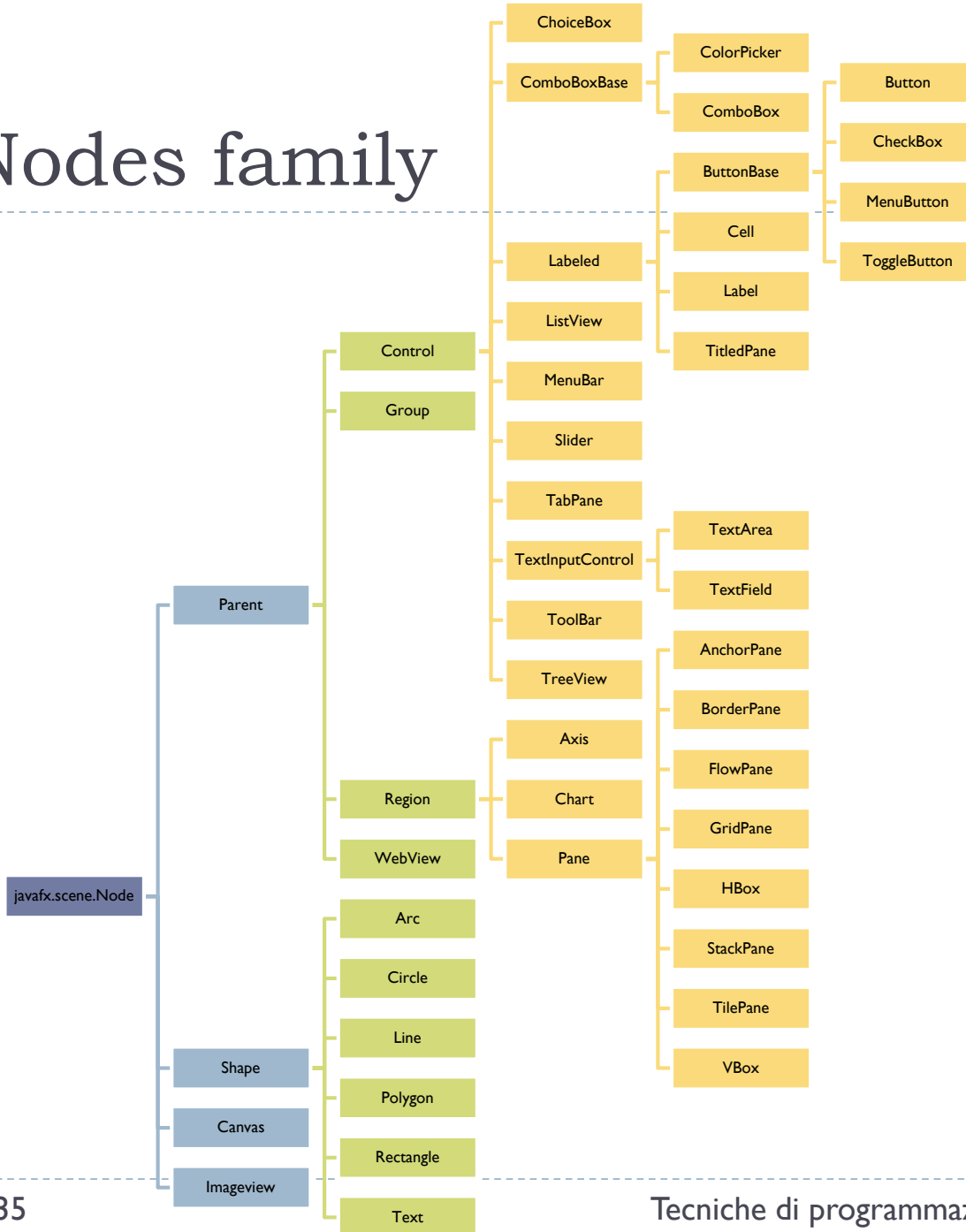
---

- ▶ Root node: top level container
- ▶ Intermediate nodes:
  - ▶ Containers
  - ▶ Layout managers
  - ▶ UI Composite controls
- ▶ Leaf (terminal) nodes:
  - ▶ Shapes
  - ▶ UI Controls
- ▶ Organized as a Hierarchical tree





# Nodes family



Focus on  
Panels  
and  
Controls

JavaDoc  
is your  
friend

# Exploring Controls and Examples

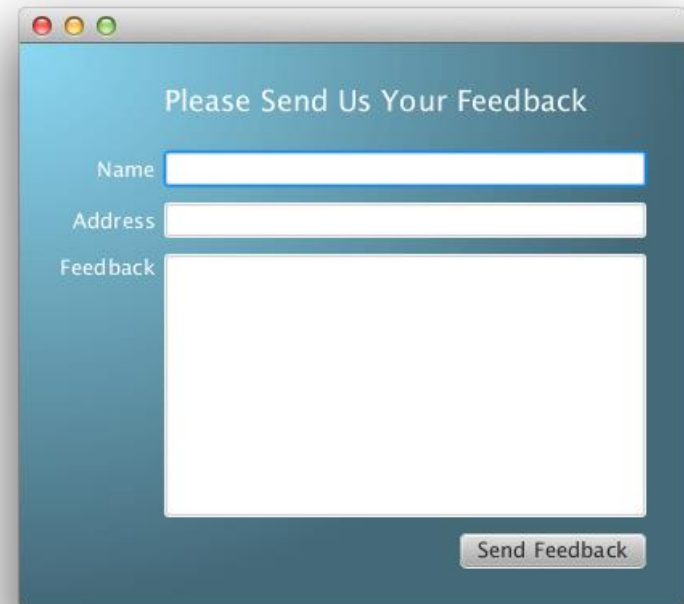
- ▶ JavaFX Ensemble demo application
- ▶ Download from Oracle site: JavaFX Demos and Samples Downloads
- ▶ Run Ensemble.jnlp



# UI Form Controls

---

- ▶ Controls may be combined to construct «Forms»
- ▶ Control Nodes have a **value** property
  - ▶ May be linked to application code
- ▶ Control Nodes generate **UI Events**
  - ▶ Button: ActionEvent
  - ▶ Text: ActionEvent, KeyTyped, KeyPressed, MouseClicked, ...



Please Send Us Your Feedback

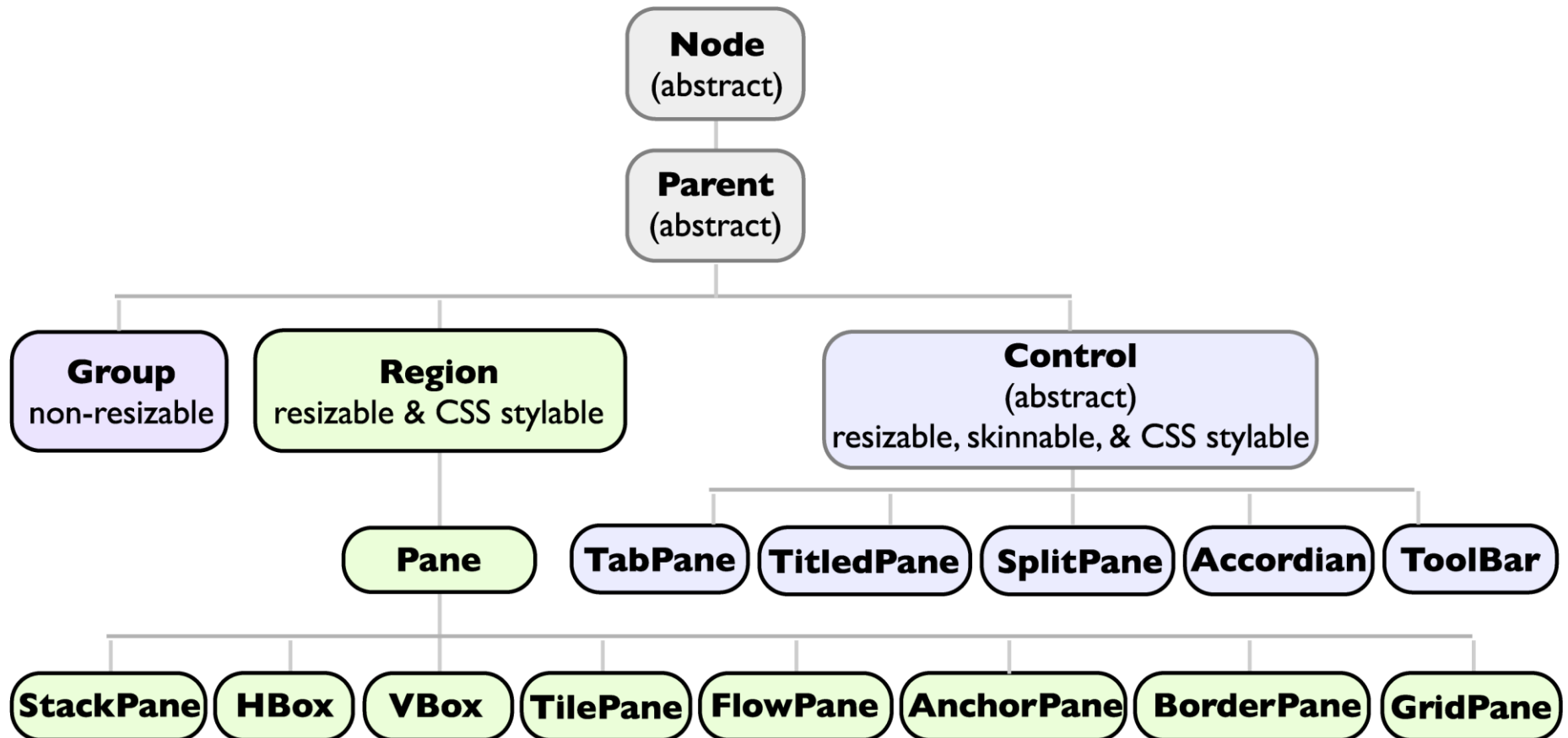
Name

Address

Feedback

Send Feedback

# JavaFX 2.0 Layout Classes



# Layout Class Hierarchy

---

- ▶ **Group:**
  - ▶ Doesn't perform any positioning of children.
  - ▶ To statically assemble a collection of nodes in fixed positions
  - ▶ To apply an effect or transform to that collection.
- ▶ **Region:**
  - ▶ base class for all general purpose layout panes
  - ▶ resizable and stylable via CSS
  - ▶ Supports dynamic layout by sizing and positioning children
- ▶ **Control:**
  - ▶ the base class for all skinnable controls
  - ▶ resizable and subclasses are all stylable via CSS
  - ▶ Controls delegate layout to their skins (which are Regions)
  - ▶ Each layout Control subclass provides API for adding content in the appropriate place within its skin
    - ▶ you do not add children to a control directly.

### Node (abstract)

```
public boolean isResizable() // returns false
public Orientation getContentBias();
public double minWidth(double height)
public double minHeight(double width)
public double prefWidth(double height)
public double prefHeight(double width)
public double maxWidth(double height)
public double maxHeight(double width)
public double getBaselineOffset()

public void relocate(double x, double y)
public void resize(double width, double height)
public void resizeRelocate(double x, double y, double w, double h)
public void autosize()
```

### Parent (abstract)

```
protected ObservableList<Node> getChildren()
public ObservableList<Node> getChildrenUnmodifiable()
```

### Group

```
isResizable() == false
public ObservableList<Node> getChildren()
public boolean isAutoSizeChildren()
public void setAutoSizeChildren(boolean v)
```

### Region

```
isResizable() == true
public Insets getPadding()
public void setPadding(Insets p)

public void setMinWidth(double w)
public double getMinWidth()
public void setMinHeight(double h)
public double getMinHeight()
public void setPrefWidth(double w)
public double getPrefWidth()
public void setPrefHeight(double h)
public double getPrefHeight()
public void setMaxWidth(double w)
public double getMaxWidth()
public void setMaxHeight(double h)
public double getMaxHeight()

public void setMinSize(double w, double h)
public void setPrefSize(double w, double h)
public void setMaxSize(double w, double h)
```

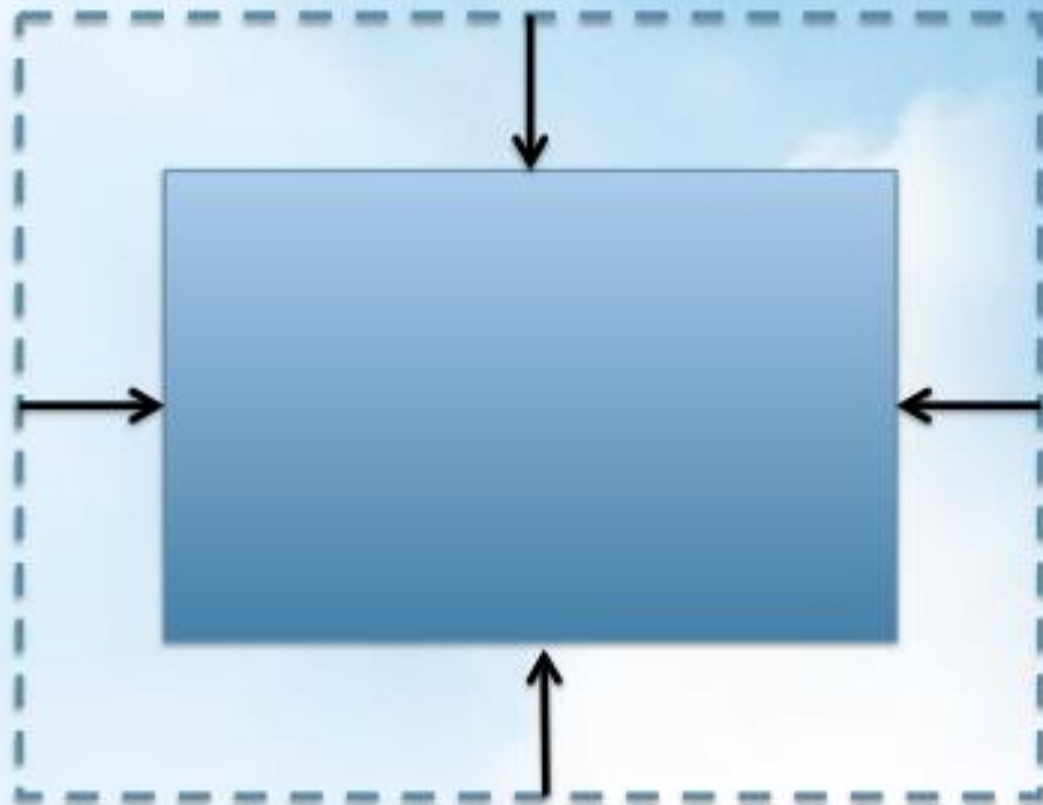
### Control (abstract)

```
isResizable() == true
public void setMinWidth(double w)
public double getMinWidth()
public void setMinHeight(double h)
public double getMinHeight()
public void setPrefWidth(double w)
public double getPrefWidth()
public void setPrefHeight(double h)
public double getPrefHeight()
public void setMaxWidth(double w)
public double getMaxWidth()
public void setMaxHeight(double h)
public double getMaxHeight()

public void setMinSize(double w, double h)
public void setPrefSize(double w, double h)
public void setMaxSize(double w, double h)
```

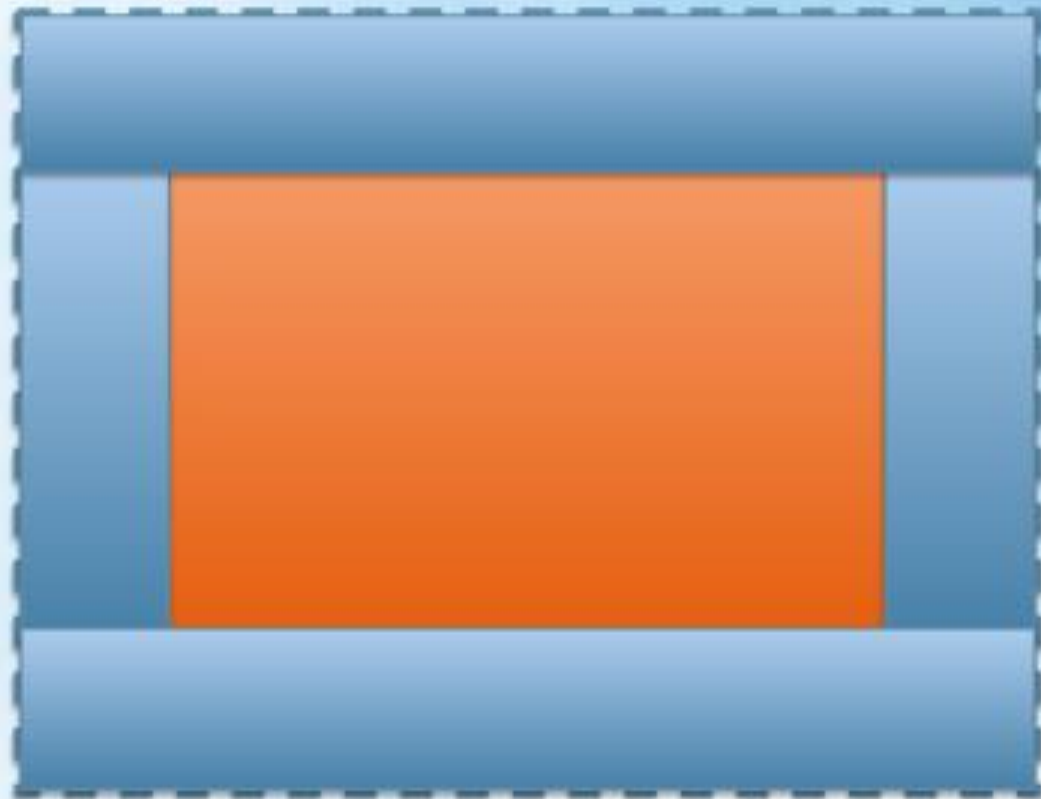
# Built-in Layouts

- > **AnchorPane**
- > BorderPane
- > VBox/HBox
- > FlowPane
- > StackPane
- > TilePane
- > GridPane



# Built-in Layouts

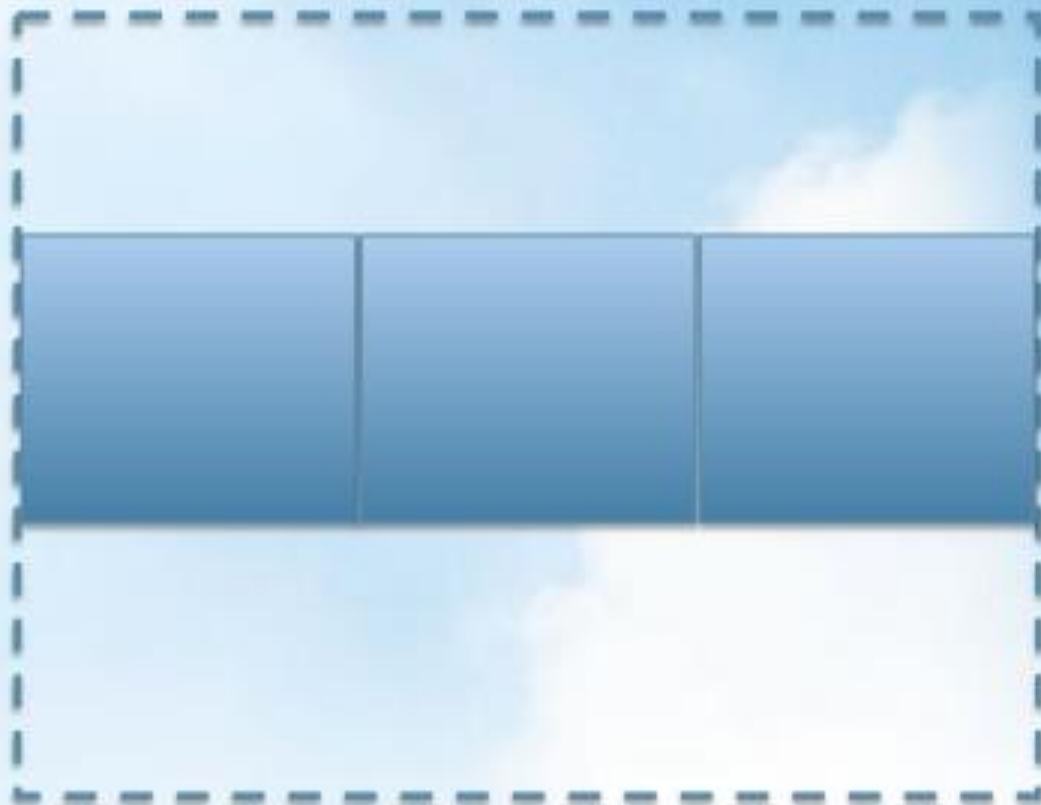
- > AnchorPane
- > **BorderPane**
- > VBox/HBox
- > FlowPane
- > StackPane
- > TilePane
- > GridPane





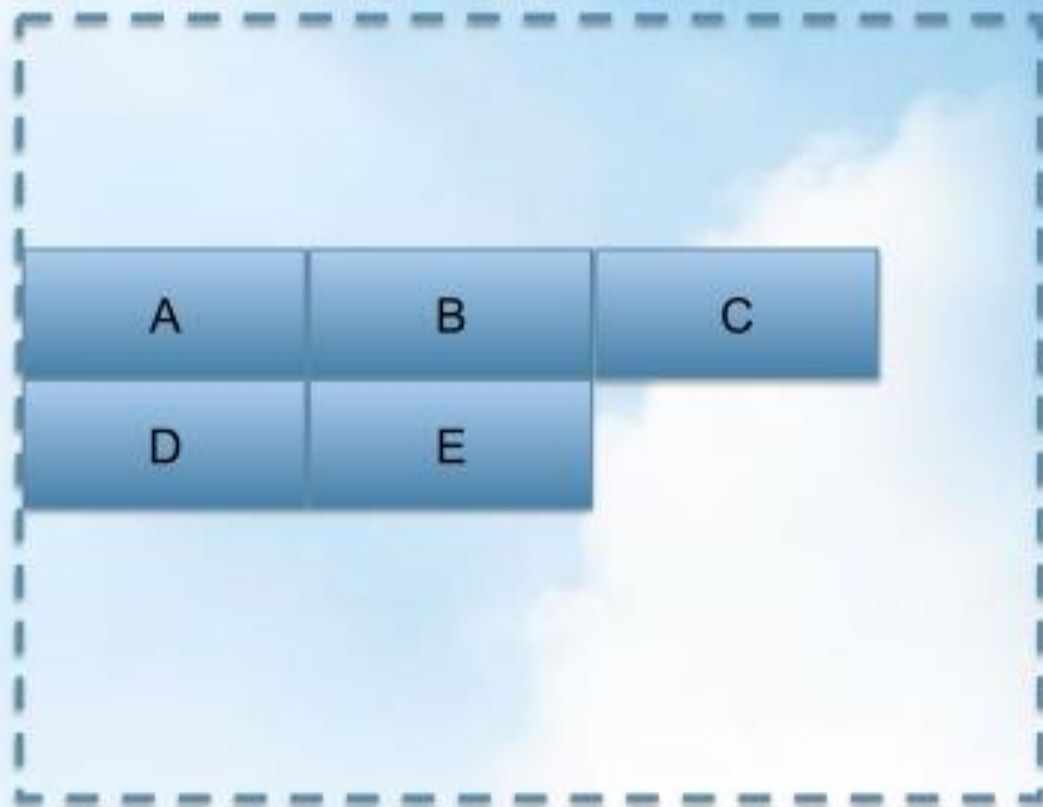
# Built-in Layouts

- > AnchorPane
- > BorderPane
- > **VBox/HBox**
- > FlowPane
- > StackPane
- > TilePane
- > GridPane



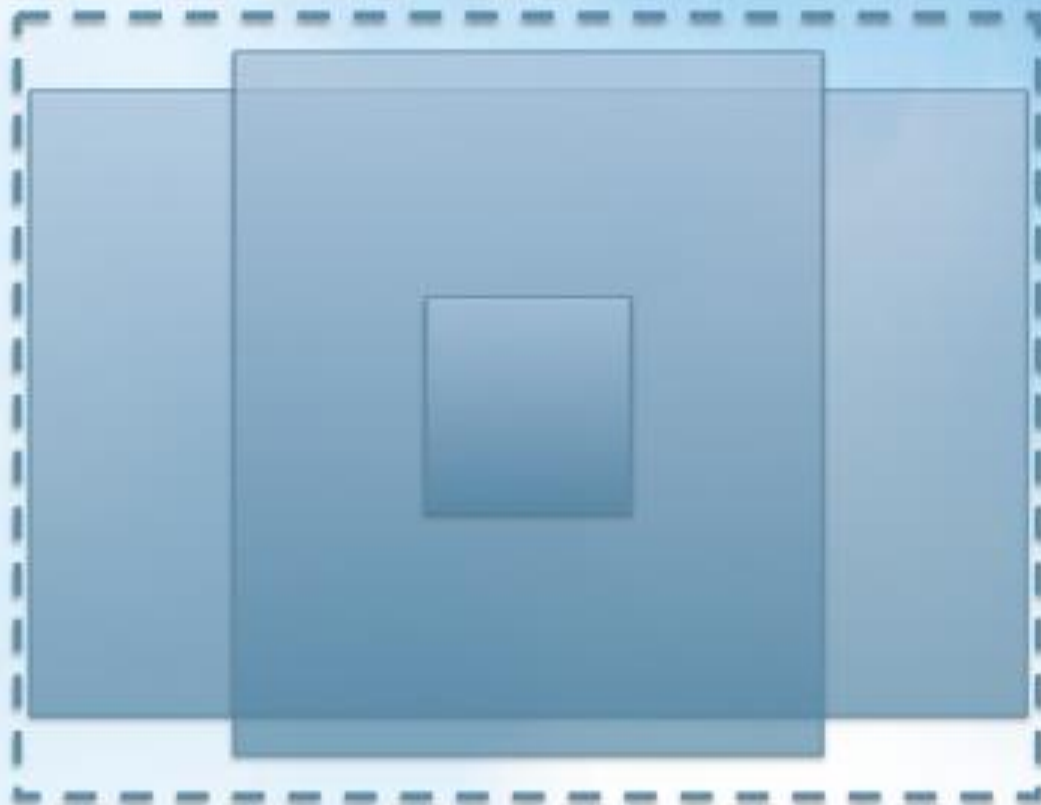
# Built-in Layouts

- > AnchorPane
- > BorderPane
- > VBox/HBox
- > **FlowPane**
- > StackPane
- > TilePane
- > GridPane



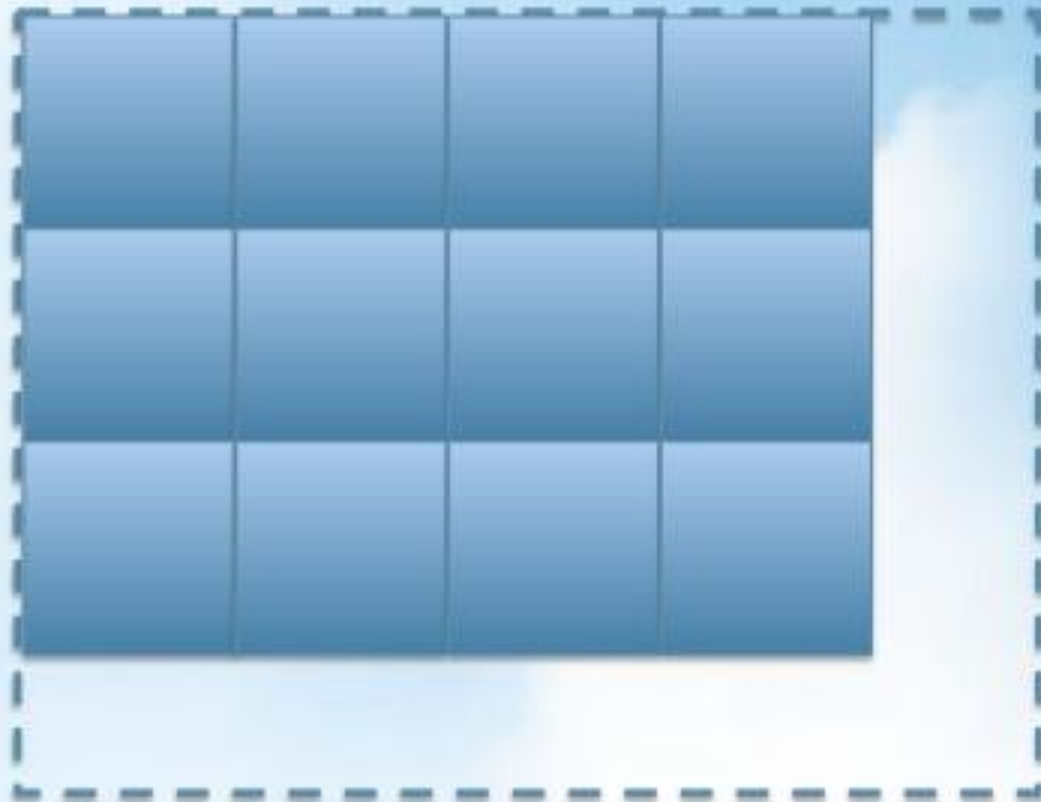
# Built-in Layouts

- > AnchorPane
- > BorderPane
- > VBox/HBox
- > FlowPane
- > **StackPane**
- > TilePane
- > GridPane



# Built-in Layouts

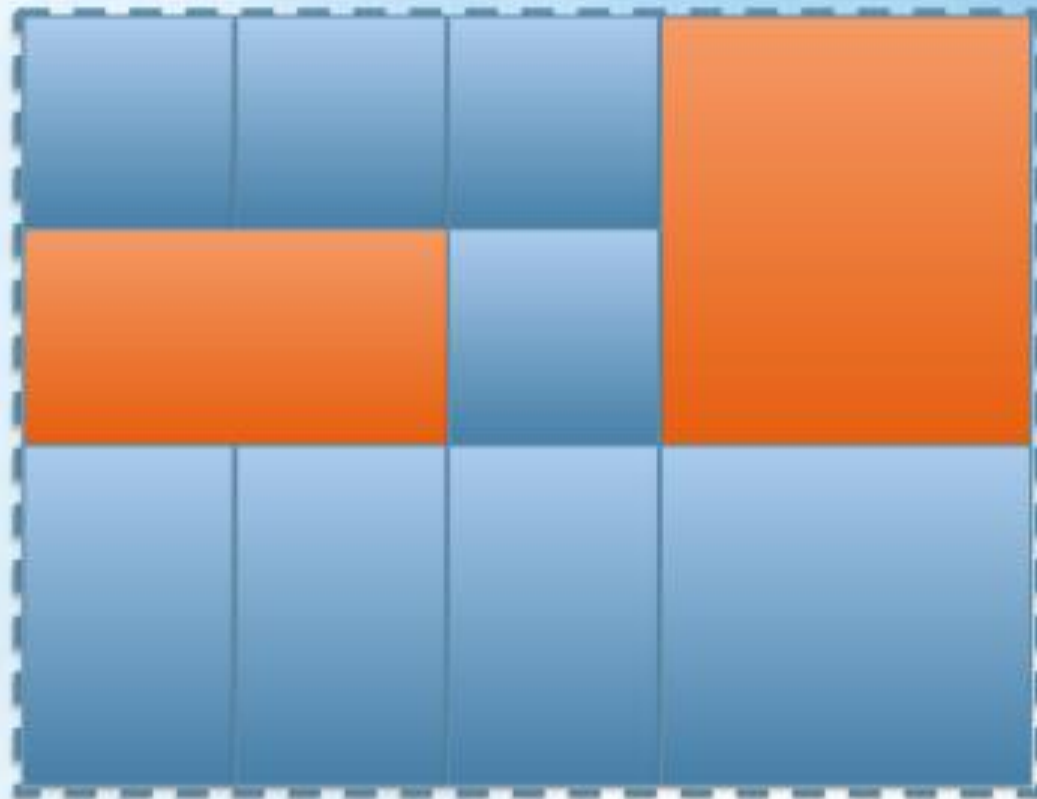
- > AnchorPane
- > BorderPane
- > VBox/HBox
- > FlowPane
- > StackPane
- > **TilePane**
- > GridPane





# Built-in Layouts

- > AnchorPane
- > BorderPane
- > VBox/HBox
- > FlowPane
- > StackPane
- > TilePane
- > **GridPane**



# Creating the Scene Graph

---

## ▶ The Java way

- ▶ Create Control Nodes
- ▶ Set properties to new nodes
- ▶ Add new nodes to parent node
- ▶ With Constructors and/or with Builders

## ▶ The FXML way

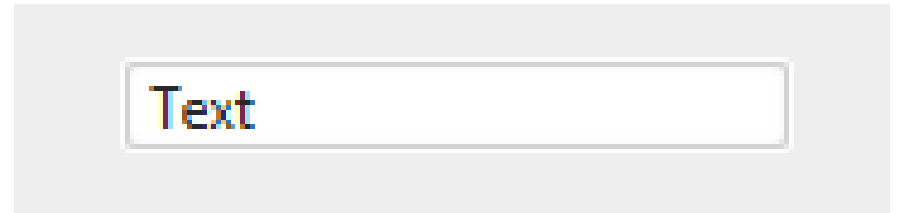
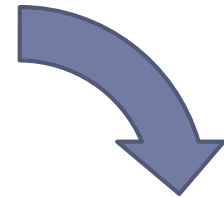
- ▶ Create a FXML file
- ▶ Define Nodes and Properties in FXML
- ▶ Load the FXML
- ▶ (Optionally, add new nodes/properties the Java way)

# Example: one text input field

---

```
TextField text = new TextField("Text");  
text.setMaxSize(140, 20);  
root.getChildren().add(text);
```

Constructors



```
TextField text = TextFieldBuilder().create()  
    .maxHeight(20).maxWidth(140)  
    .text("Text")  
    .build() ;
```



Builders

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

```

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

    @Override
    public void start(Stage primaryStage)
    {
        primaryStage.setTitle("Hello Devoxx");
        Group root = new Group();
        Scene scene = new Scene(root, 400, 250,
                                Color.ALICEBLUE);
        Text text = new Text();
        text.setX(105);
        text.setY(120);
        text.setFont(new Font(30));
        text.setText("Hello Devoxx");
        root.getChildren().add(text);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}

```



```
public void start(Stage primaryStage)
{
    primaryStage.setTitle("Hello Devovx");
    primaryStage.setScene(SceneBuilder.create()
        .width(400).height(250).fill(Color.ALICEBLUE)
        .root(GroupBuilder.create().children(
            TextBuilder.create()
                .x(105).y(120)
                .text("Hello Devovx")
                .font(new Font(30)).build()
            ).build()
        ).build());

    primaryStage.show();
}
```

# The FXML way...

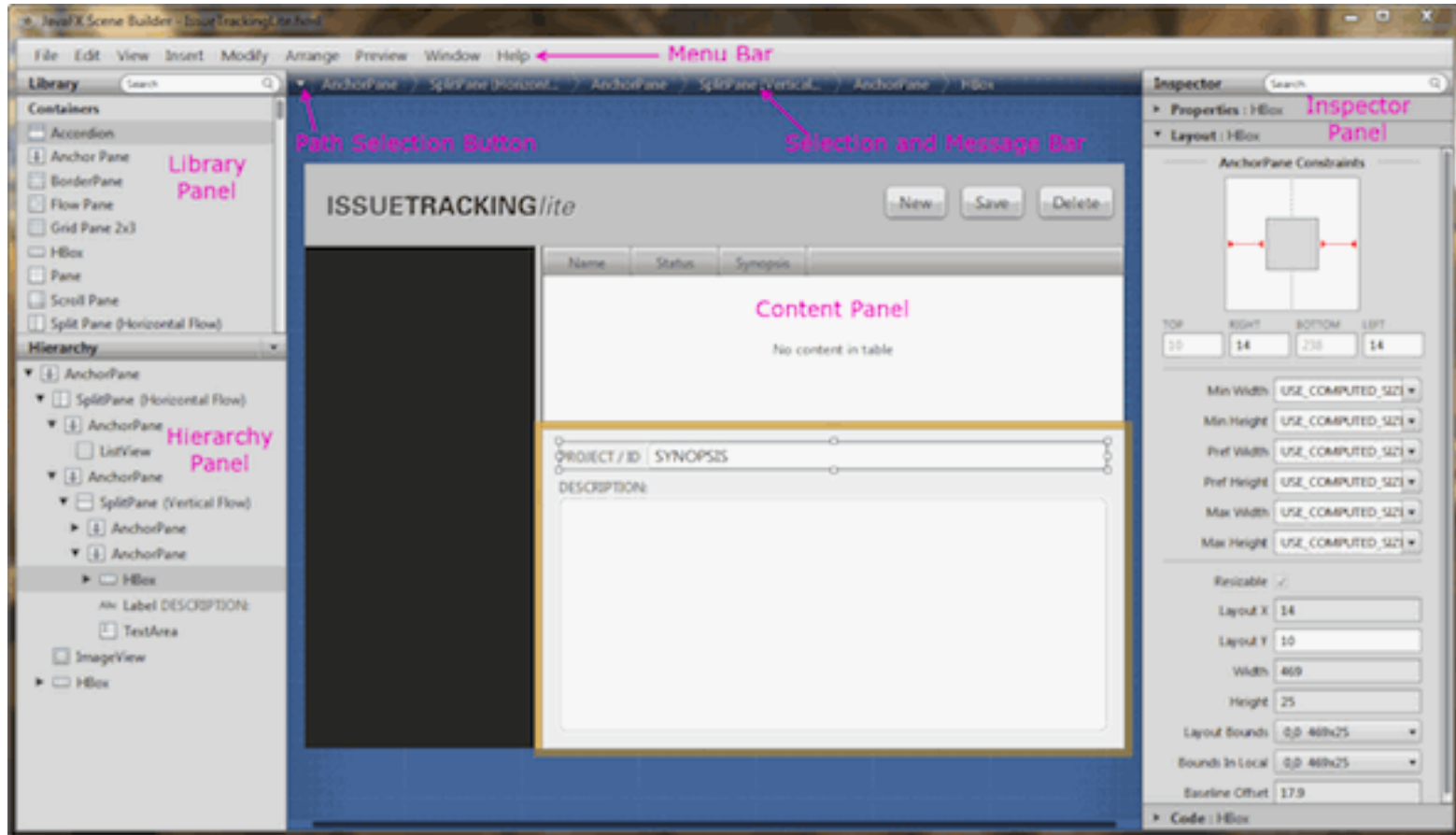
---

- ▶ XML-based format
- ▶ Nested tree of XML Elements, corresponding to Nodes
- ▶ XML Attributes corresponding to (initial) properties of nodes
- ▶ JavaFX Scene Builder is a GUI for creating FXML files
- ▶ The FXMLLoader class reads a FXML file and creates all the Nodes

# Example

```
<AnchorPane>
  <children>
    <VBox spacing="10.0"
      AnchorPane.bottomAnchor="0.0"
      AnchorPane.leftAnchor="0.0"
      AnchorPane.rightAnchor="0.0"
      AnchorPane.topAnchor="0.0">
      <children>
        <TextField fx:id="searchField"
          minHeight="-Infinity"
          onKeyTyped="#handleSearchBoxTyped"
          promptText="Search" />
        <HBox spacing="10.0">
          <children>
            <ToggleButton fx:id="toggleSession"
              selected="true"
              text="Session">
              <toggleGroup>
                <ToggleGroup fx:id="toggleSearch" />
              </toggleGroup>
            </ToggleButton>
```

# JavaFX Scene Builder



# FXMLLoader

---

```
@Override
public void start(Stage stage) throws Exception {
    Parent root = FXMLLoader.load(
        getClass().getResource("fxml_example.fxml"));

    stage.setTitle("FXML Welcome");
    stage.setScene(new Scene(root, 300, 275));
    stage.show();
}
```

# Linking FXML and Java

---

- ▶ FXML element may have an associated attribute `fx:id`
- ▶ Nodes may be later retrieved by
  - ▶ `public Node lookup(java.lang.String selector)`
  - ▶ Finds a node with a specified ID in the current sub-tree
  - ▶ Example:
    - ▶ `scene.lookup("#myId");`
- ▶ Node references can also be «injected» using the `@FXML` annotation (see later)



# Interacting with Nodes

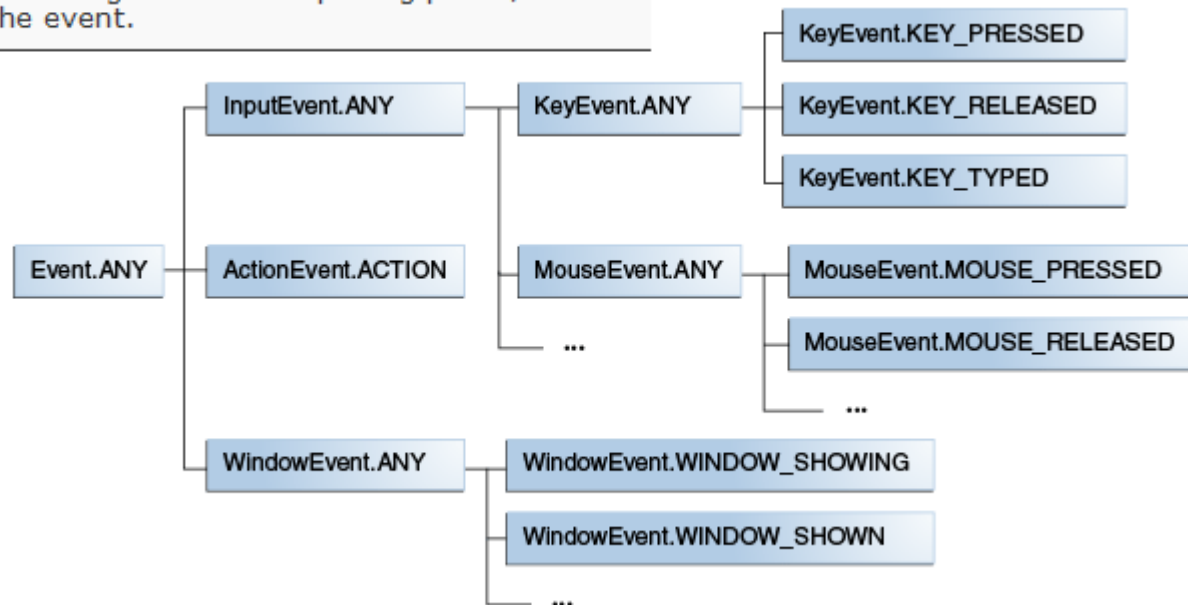
---

- ▶ In JavaFX applications, events are **notifications** that something has happened.
  - ▶ An event represents an occurrence of something of interest to the application
  - ▶ As a user clicks a button, presses a key, moves a mouse, or performs other actions, events are dispatched.
- ▶ Registered event filters and **event handlers** within the application
  - ▶ **receive** the event and
  - ▶ **provide** a response.



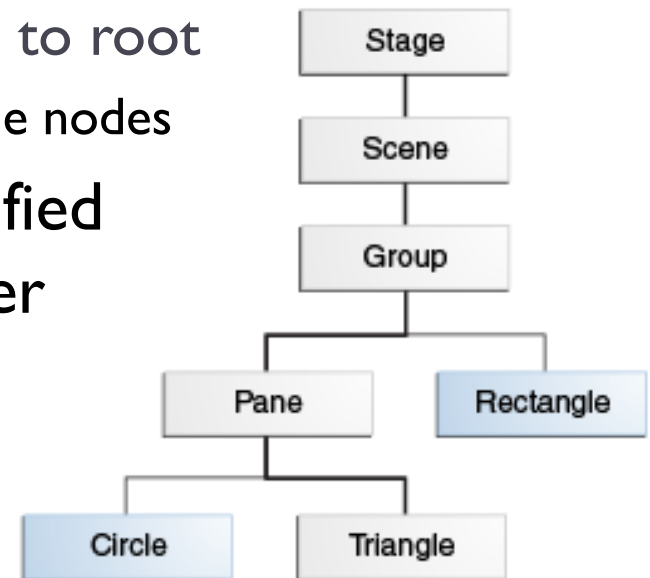
# What is an event?

Property	Description
Event type	Type of event that occurred.
Source	Origin of the event, with respect to the location of the event in the event dispatch chain. The source changes as the event is passed along the chain.
Target	Node on which the action occurred and the end node in the event dispatch chain. The target does not change, however if an event filter consumes the event during the event capturing phase, the target will not receive the event.



# Event propagation

- ▶ Events are generated on the source node
- ▶ Events propagated in the scene graph hierarchy («dispatch chain»), in two phases
  - ▶ **Dispatching**: downwards, from root to source node
    - ▶ Processes Event Filters registered in the nodes
  - ▶ **Bubbling**: upwards, from source node to root
    - ▶ Processes Event Handlers registered in the nodes
- ▶ If you want an application to be notified when an event occurs, register a filter or a handler for the event
- ▶ Handlers may “consume” the event



# Event Handlers

---

- ▶ Implements the EventHandler interface
- ▶ Executed during the event bubbling phase.
- ▶ If does not consume the event, it is propagated to the parent.
- ▶ A node can register more than one handler.
- ▶ Handlers for a specific event type are executed before handlers for generic event types.
  - ▶ For example, a handler for the `KeyEvent.KEY_TYPED` event is called before the handler for the `InputEvent.ANY` event.
- ▶ To consume an event, call the `consume()` method

# Registering Event Handlers

---

- ▶ `setOnEvent-type(  
EventHandler<? super event-class> value )`
  - ▶ **Event-Type**
    - ▶ The type of event that the handler processes (e.g. `setOnKeyTyped`, `setOnMouseClicked`, ...)
  - ▶ **Event-class**
    - ▶ The class that defines the event type (e.g., `KeyEvent` , `MouseEvent`, ...)
  - ▶ **Value**
    - ▶ The event handler for event-class (or for one of its super classes)
    - ▶ Must implement: `public void handle(ActionEvent event)`
    - ▶ May be a regular class or an anonymous inline class

# Example

```
class ButtonActionHandler implements
javafx.event.EventHandler<ActionEvent> {

    public ButtonActionHandler (/*params*/) {
        // constructor - if needed
    }

    @Override
    public void handle(ActionEvent event) {
        Button b = (Button)event.getSource() ;
        //...do something
        String buttonText = b.getText() ;
        // ...
    }
}
```

Event Handler

Registration

```
Button btn = new Button() ;

btn.setOnAction(new ButtonActionHandler()) ;
```

# Example (inline definition)

---

Registration &  
Anonymous event handler

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