POLITECNICO DI TORINO

e-Lite

Programming with JavaFX

Tecniche di Programmazione – A.A. 2016/2017

Summary

About and History

Introduction to JavaFX

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)

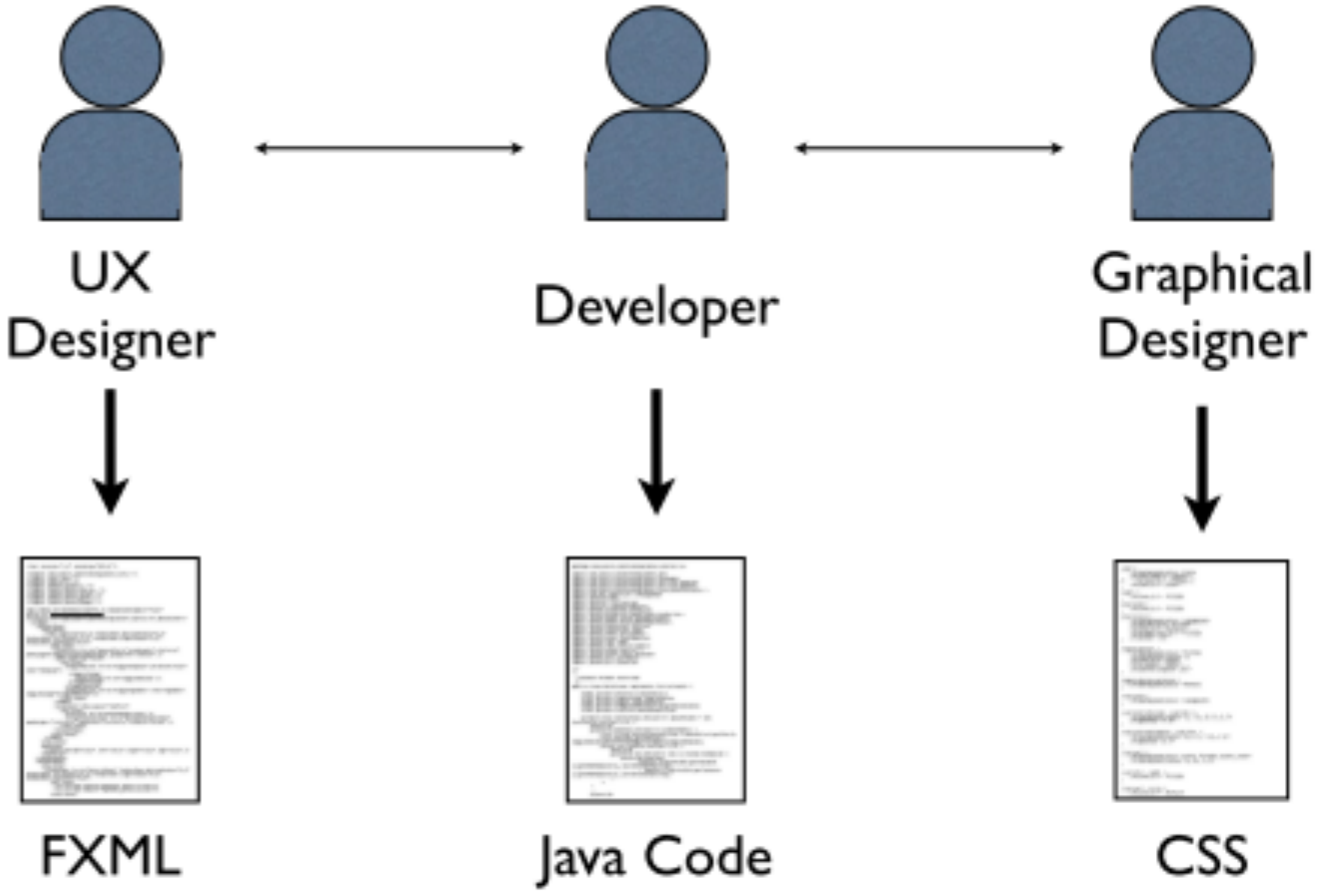
Getting and running JavaFX

- 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

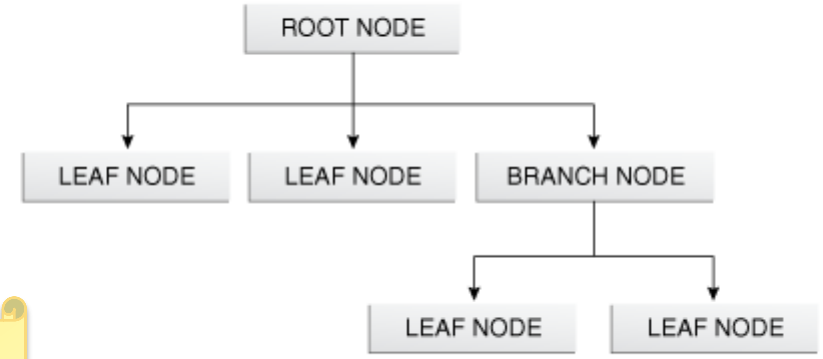


Empty JavaFX window

```
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);
    }
}
```

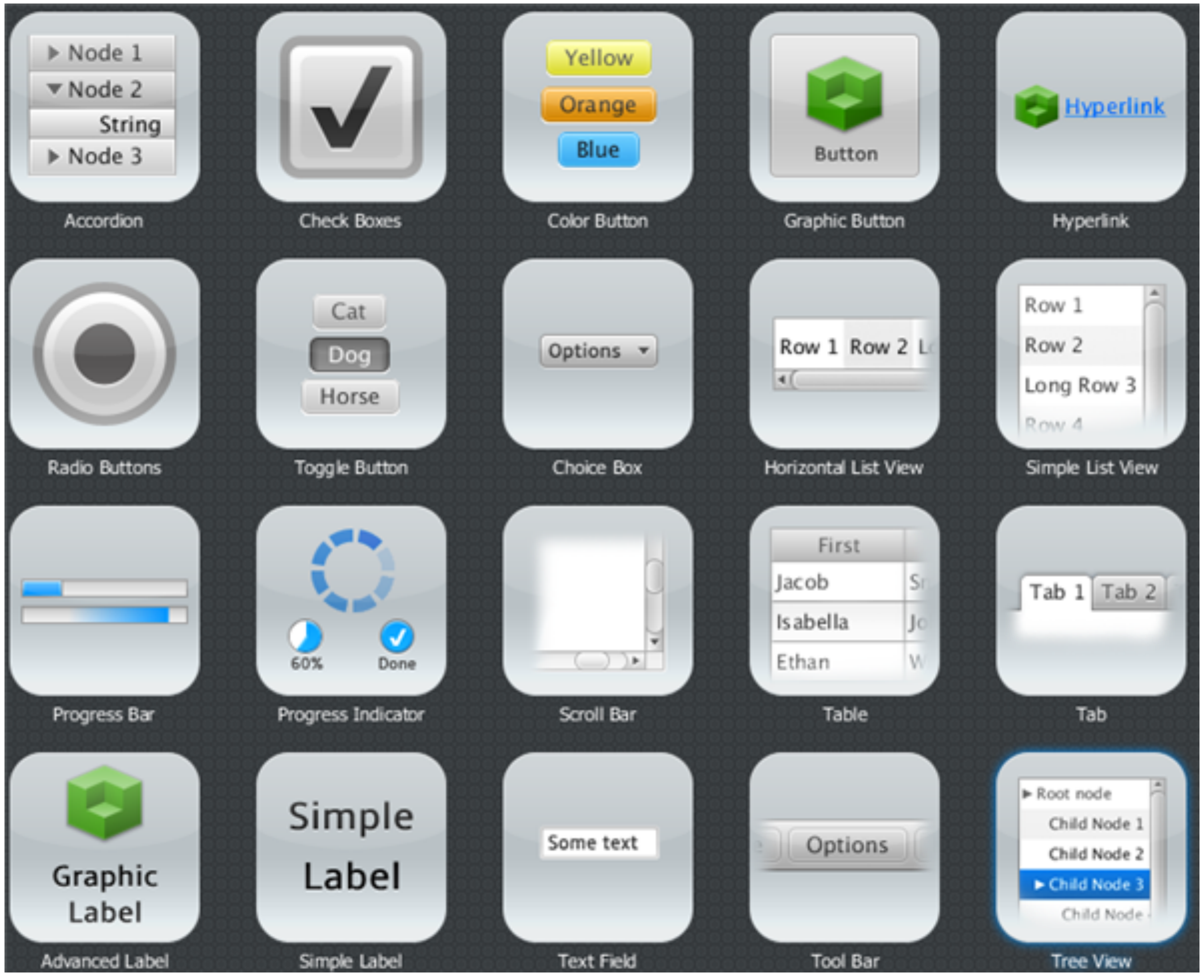
Key concepts in JavaFX

- **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



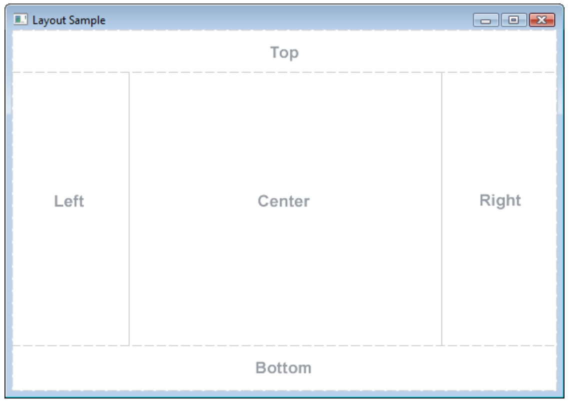
My best friend is the JavaFX JavaDoc API
http://docs.oracle.com/javase/8/javafx/api/

Some 'Leaf' Nodes (Controls)

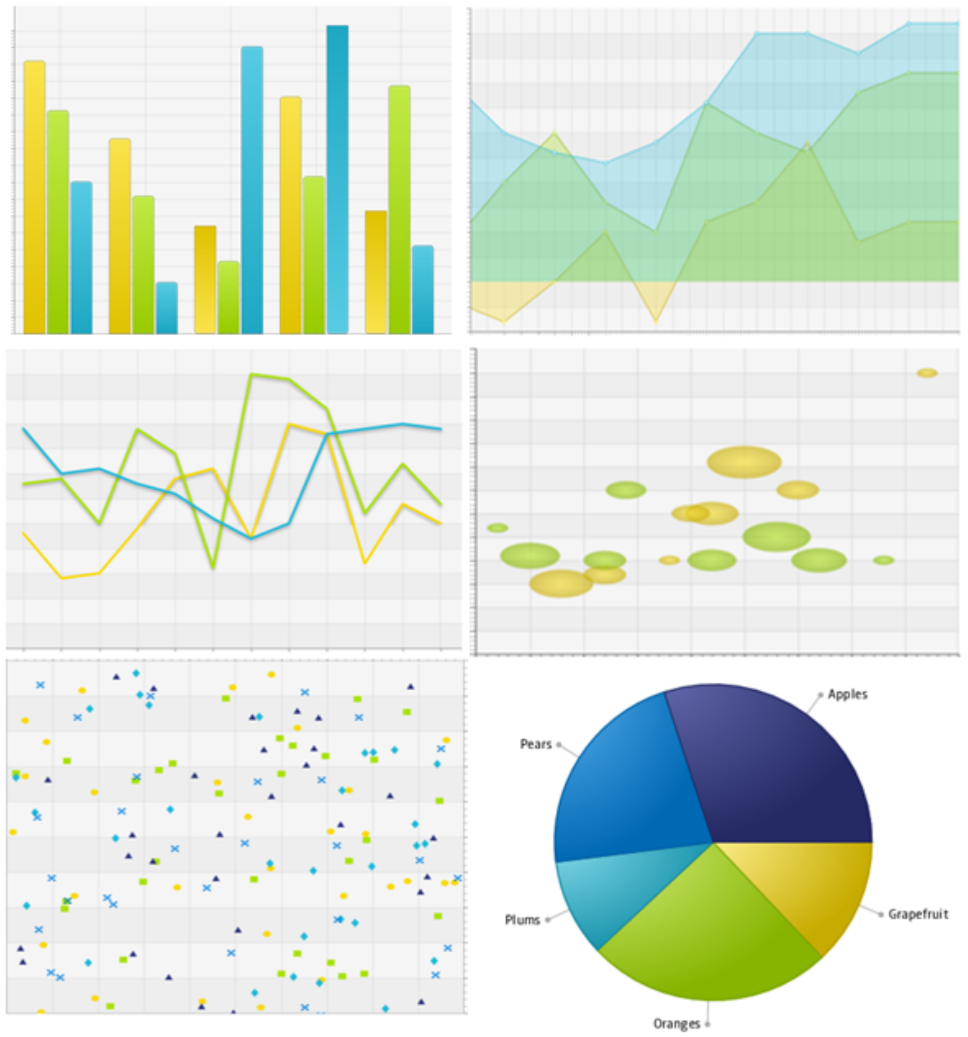


Some 'Parent' Nodes (Container 'Panes')

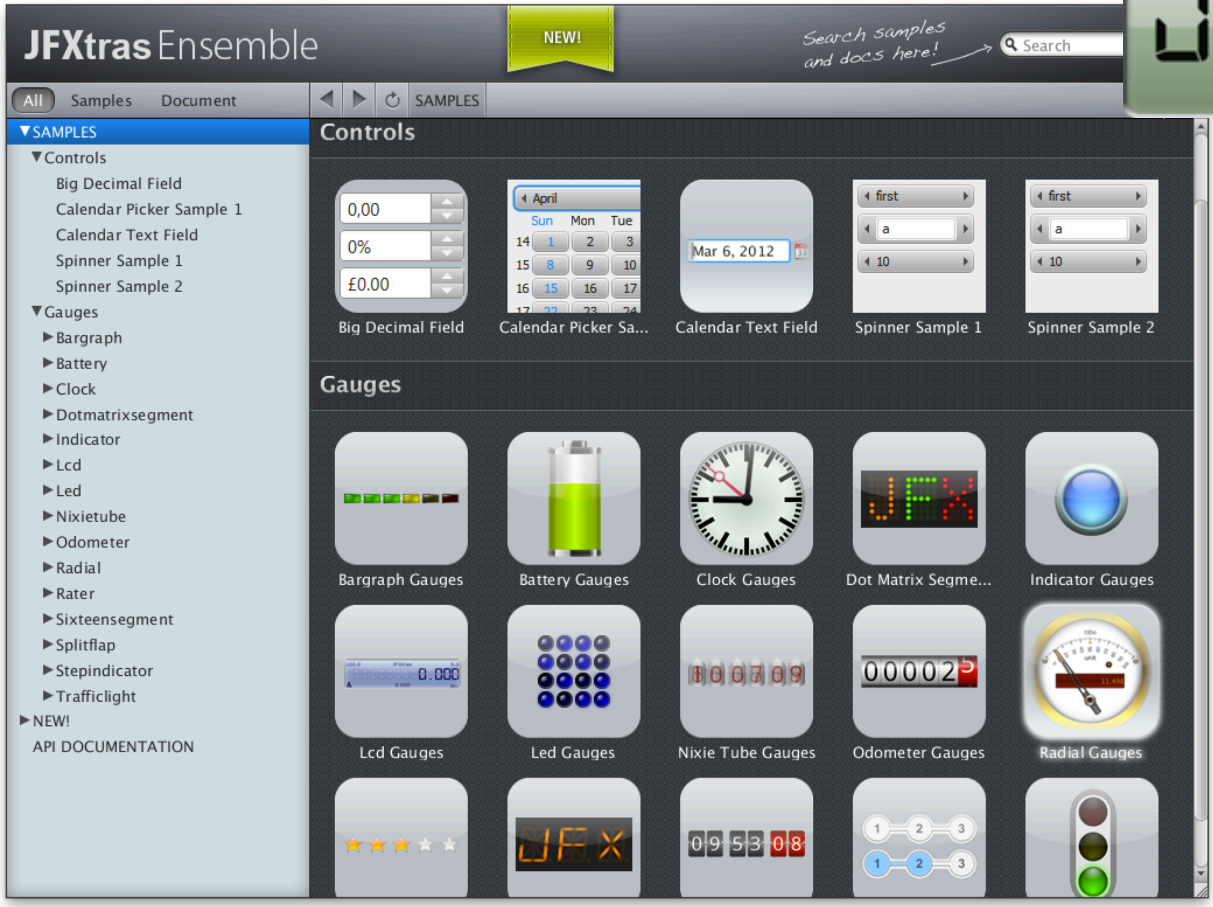
- BorderPane (5-areas)
- Hbox, Vbox (linear sequence)
- StackPane (overlay all children)
- GridPane (row x columns)
- FlowPane (flowing boxes, wrap around)
- TilePane (flowpane with equally sized boxes)
- AnchorPane (magnetically attach nodes at corners or sides)



Some Nodes (Charts)



And more coming…

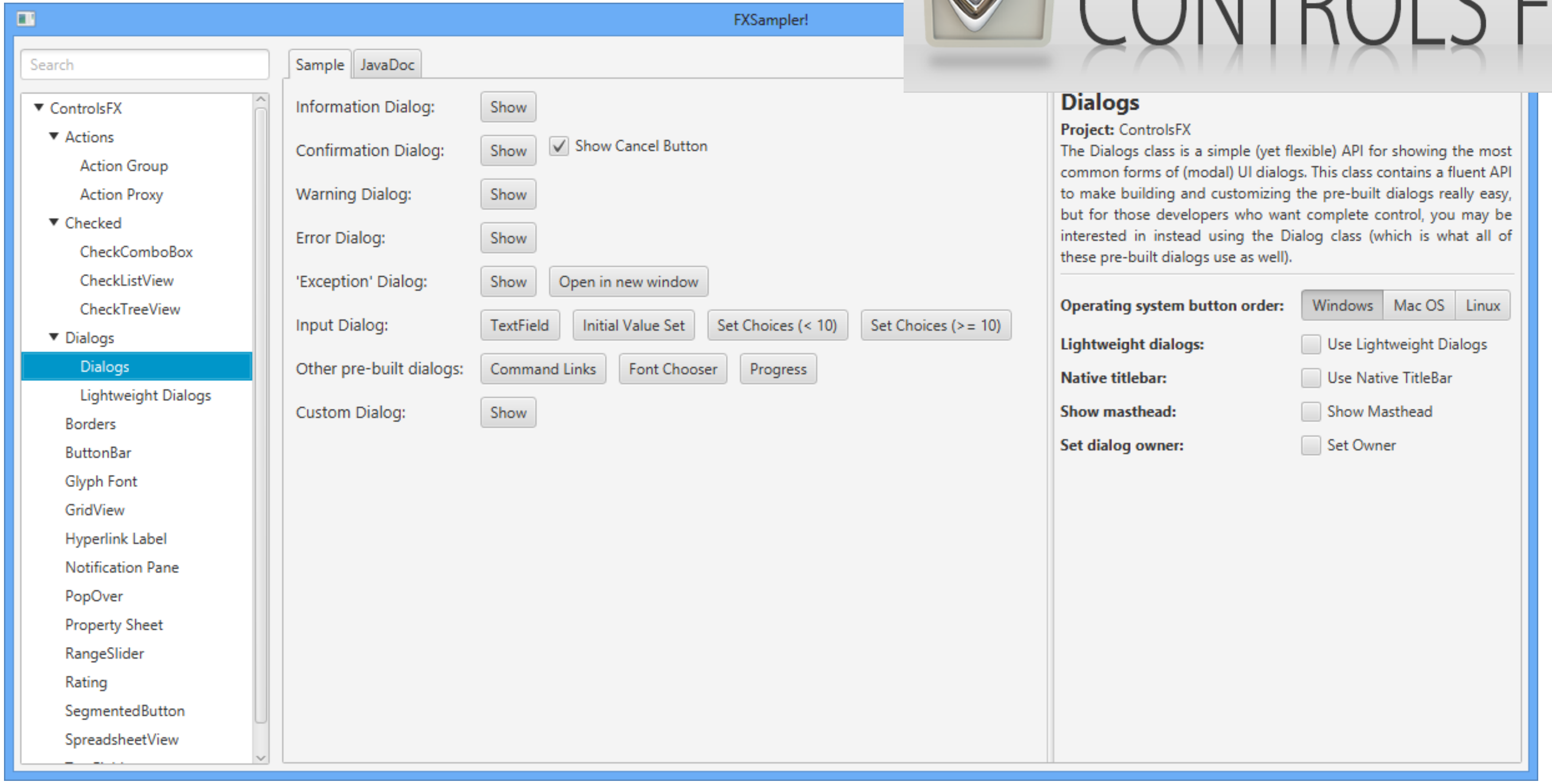


JFXTRAS ensemble

http://jfxtras.org/

And more coming…

http://fxexperience.com/controlsfx/



Information Dialog: Show
Confirmation Dialog: Show ☑ Show Cancel Button
Warning Dialog: Show
Error Dialog: Show
'Exception' Dialog: Show Open in new window
Input Dialog: TextField Initial Value Set Set Choices (< 10) Set Choices (>= 10)
Other pre-built dialogs: Command Links Font Chooser Progress
Custom Dialog: Show

Dialogs

Project: ControlsFX

The Dialogs class is a simple (yet flexible) API for showing the most common forms of (modal) UI dialogs. This class contains a fluent API to make building and customizing the pre-built dialogs really easy, but for those developers who want complete control, you may be interested in instead using the Dialog class (which is what all of these pre-built dialogs use as well).

Operating system button order: Windows Mac OS Linux
Lightweight dialogs: ☐ Use Lightweight Dialogs
Native titlebar: ☐ Use Native TitleBar
Show masthead: ☐ Show Masthead
Set dialog owner: ☐ Set Owner

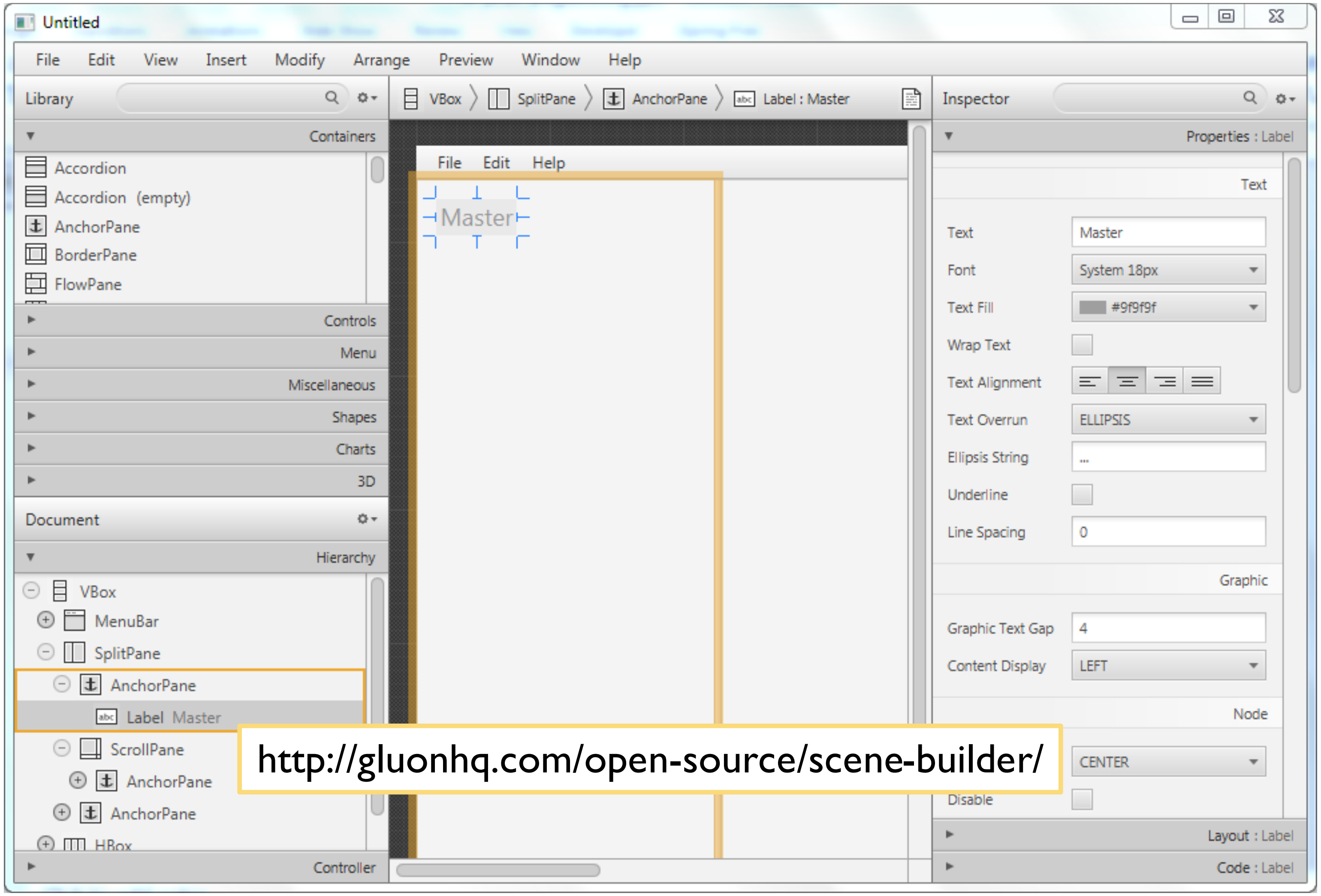
How to add scene content

- 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



Text	Master
Font	System 18px
Text Fill	#9f9f9f
Wrap Text	☐
Text Alignment	
Text Overrun	ELLIPSIS
Ellipsis String	...
Underline	☐
Line Spacing	0
Graphic Text Gap	4
Content Display	LEFT
	CENTER
Disable	☐

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();
}
```

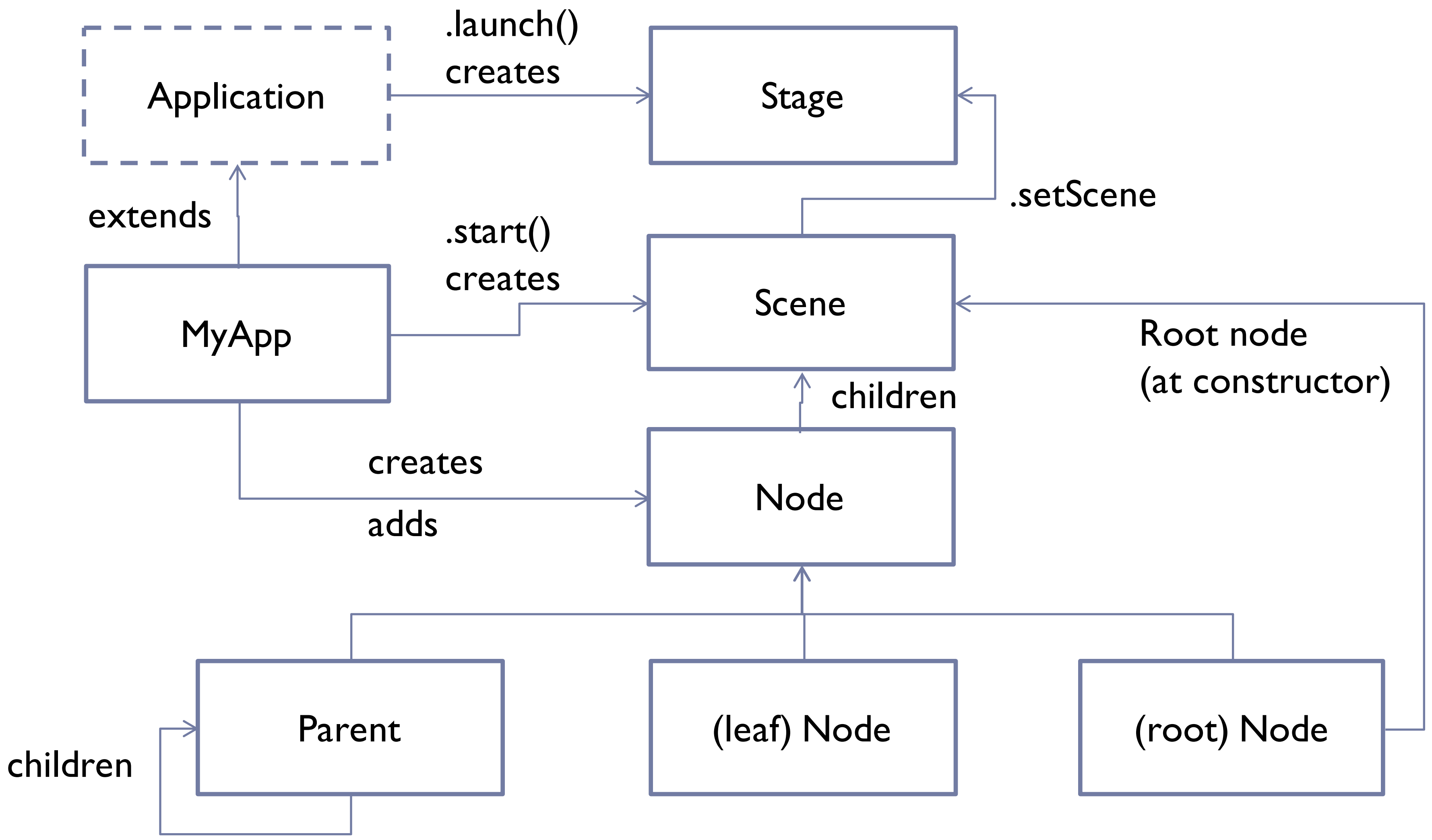
Application structure

Introduction to JavaFX

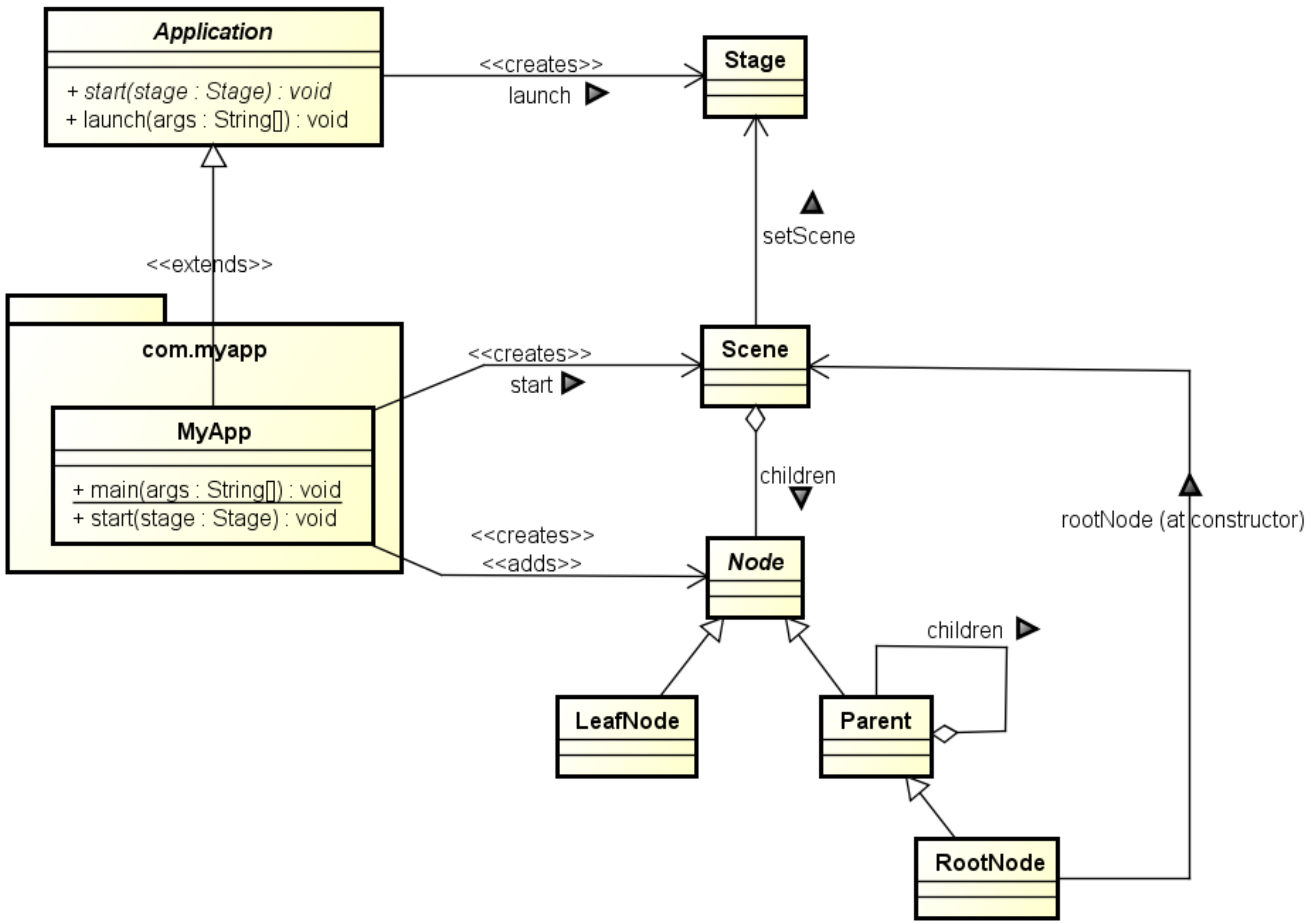
Empty JavaFX window

```
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);
    }
}
```

General class diagram



Typical Class Diagram

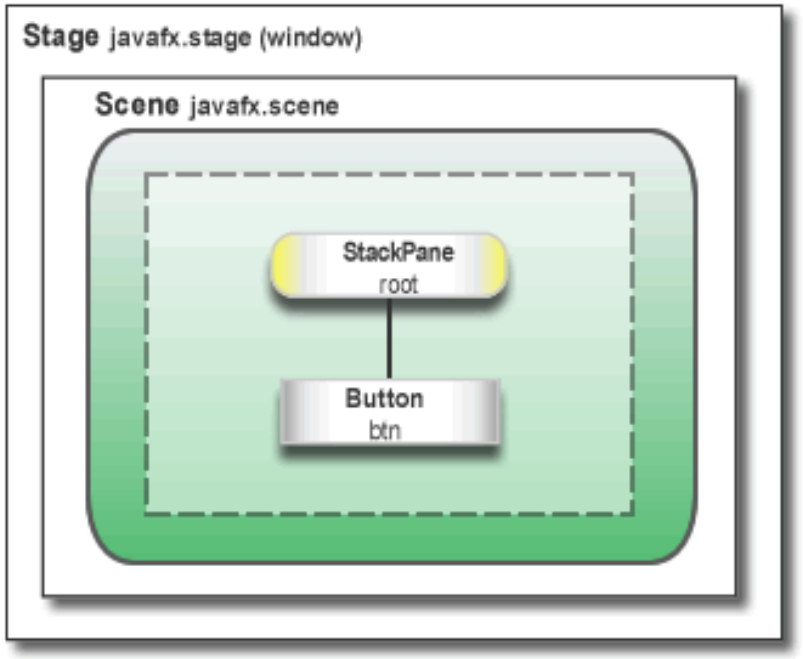


General rules

- 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

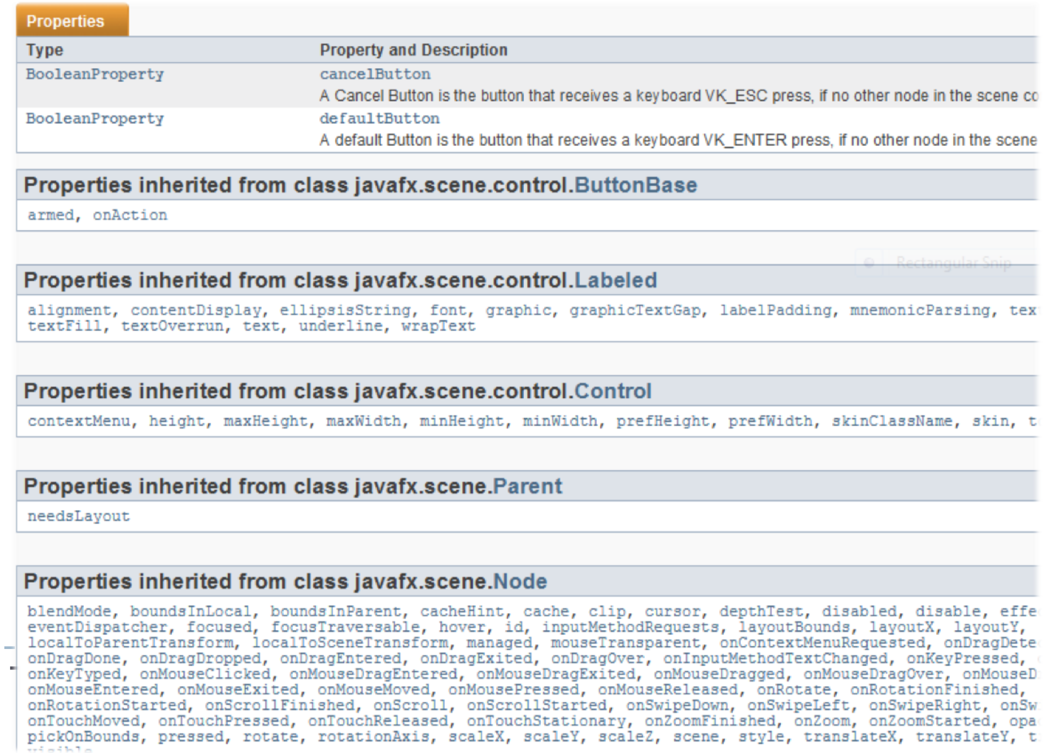
- The Scene is populated with a tree of Nodes
  - Layout components
  - UI Controls
  - Charts
  - Shapes
- Nodes have Properties
  - Visual (size, position, z-order, color, ...)
  - Contents (text, value, data sets, ...)
  - Programming (event handlers, controller)
- Nodes generate Events
  - UI events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

Bindings

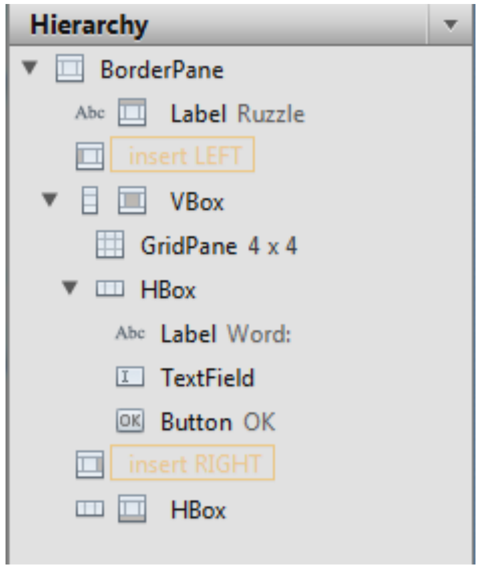
- Automatically connect («bind») one Property to another Property
  - Whenever the source property changes, the bound one is automatically updated
  - Multiple bindings are supported
  - Lazy evaluation is supported
  - Bindings may also involve computations (arithmetic operators, if-then-else, string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

The Scene graph

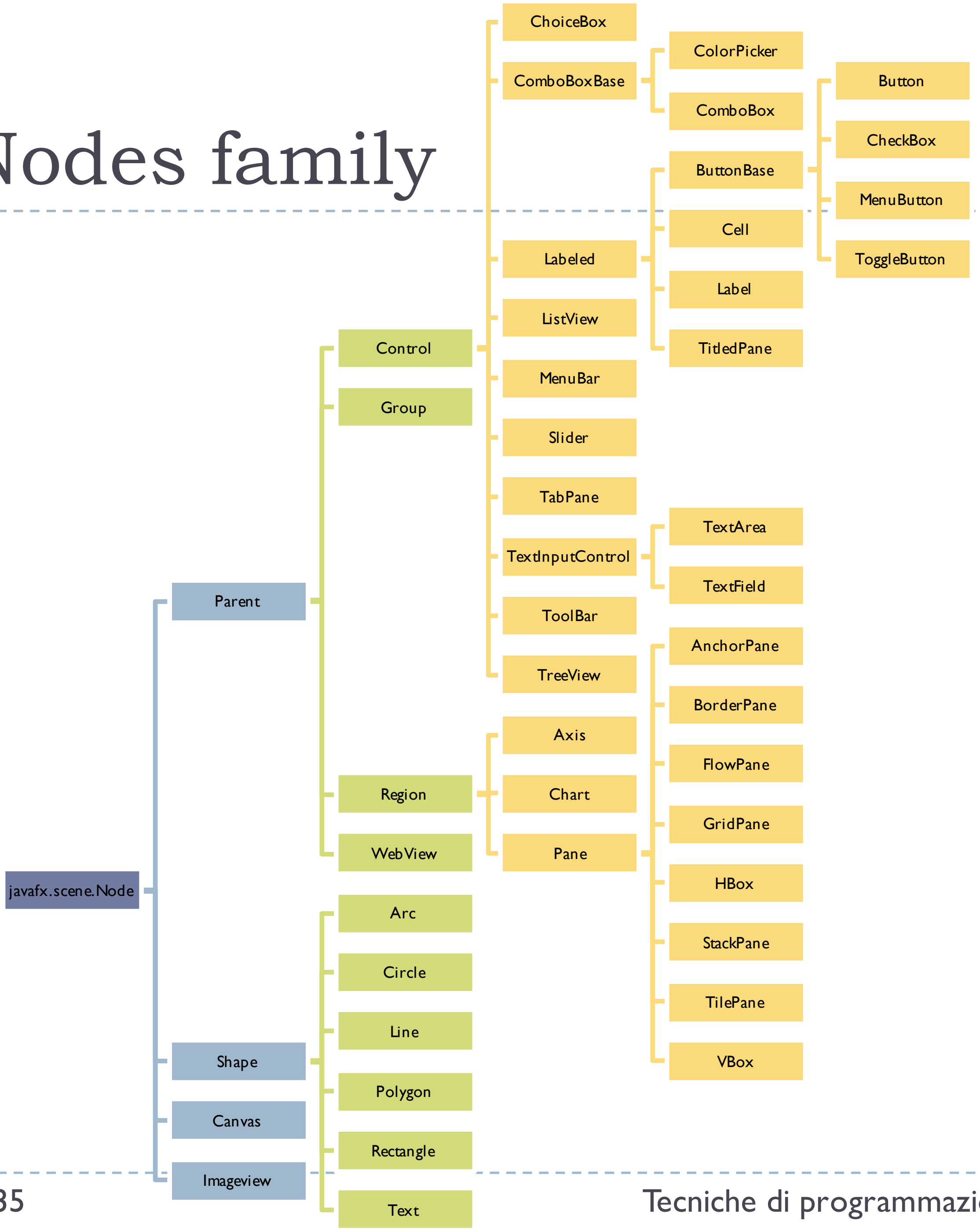
Introduction to JavaFX

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

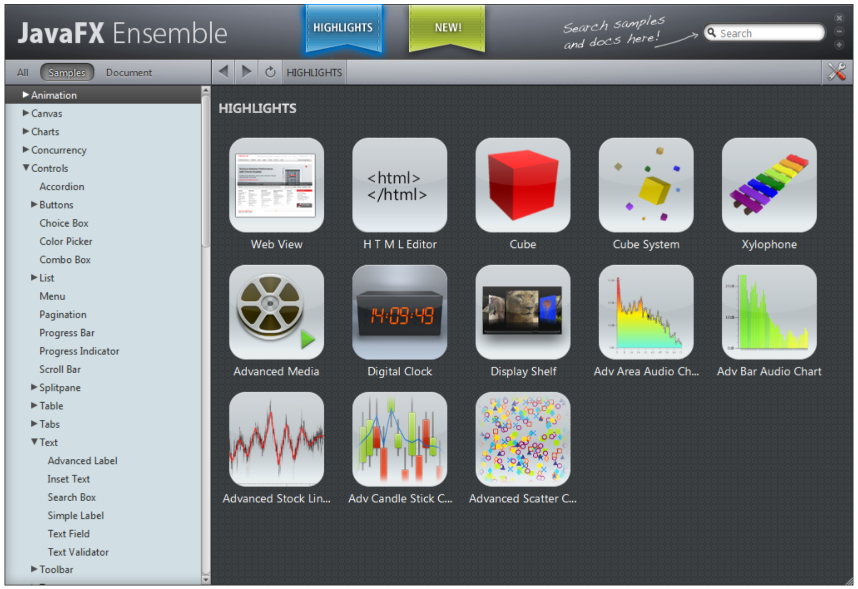


Focus on Panes and Controls

JavaDoc is your friend

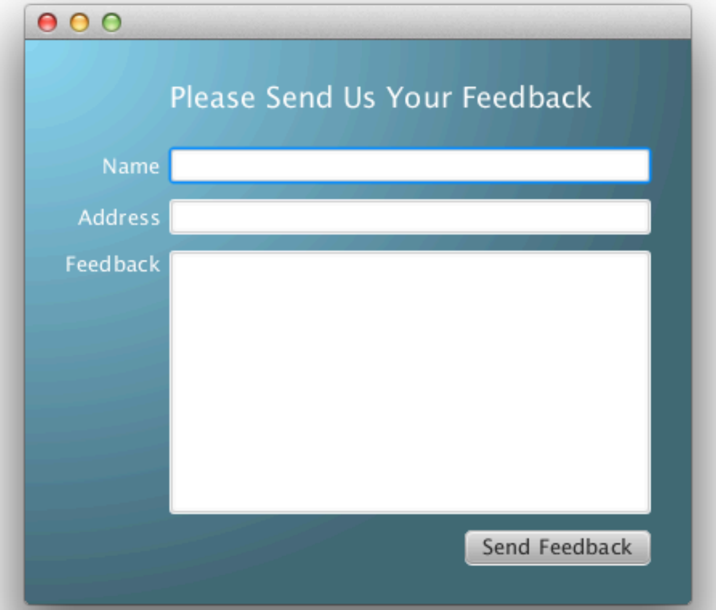
Exploring Controls and Examples

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

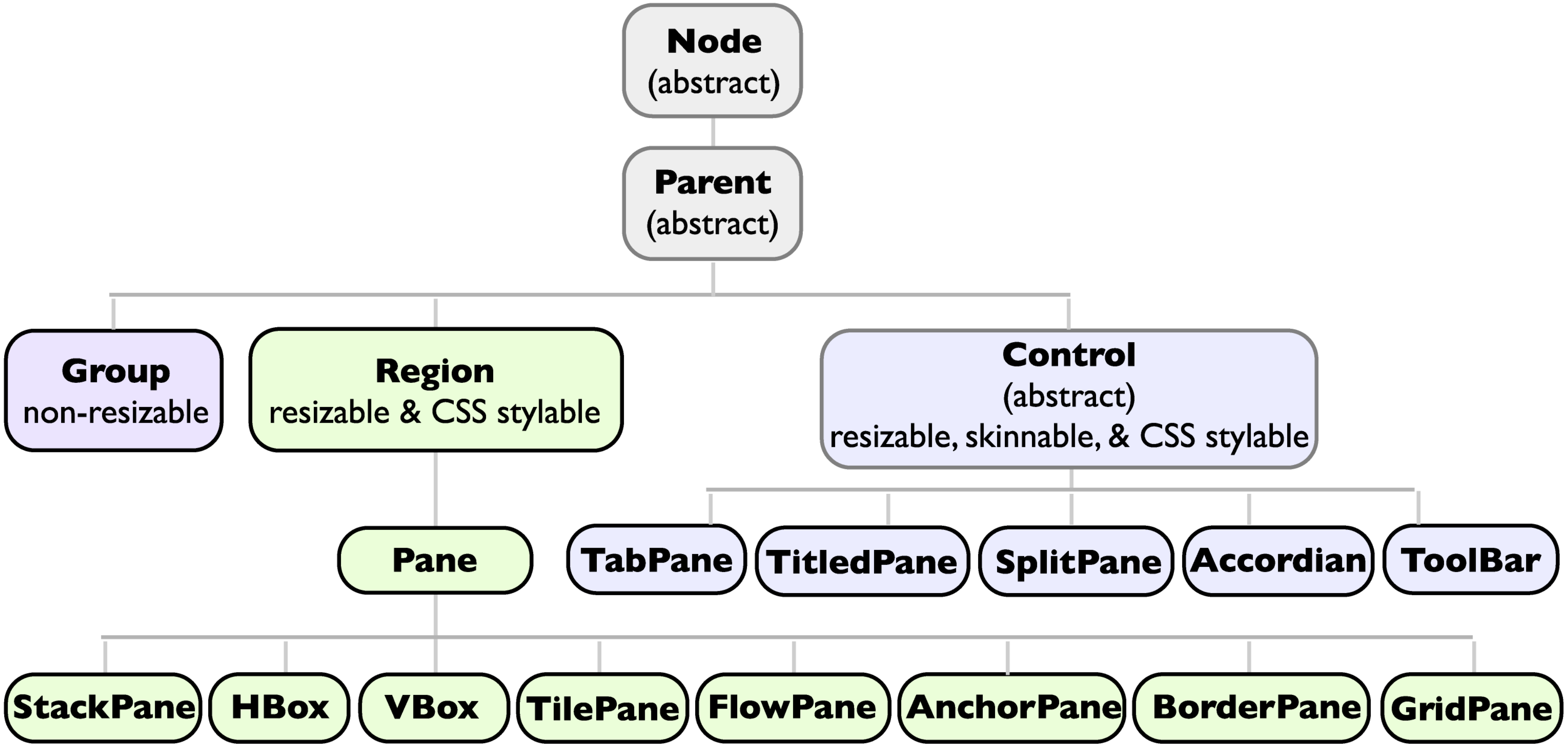


UI Form Controls

- 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, ...

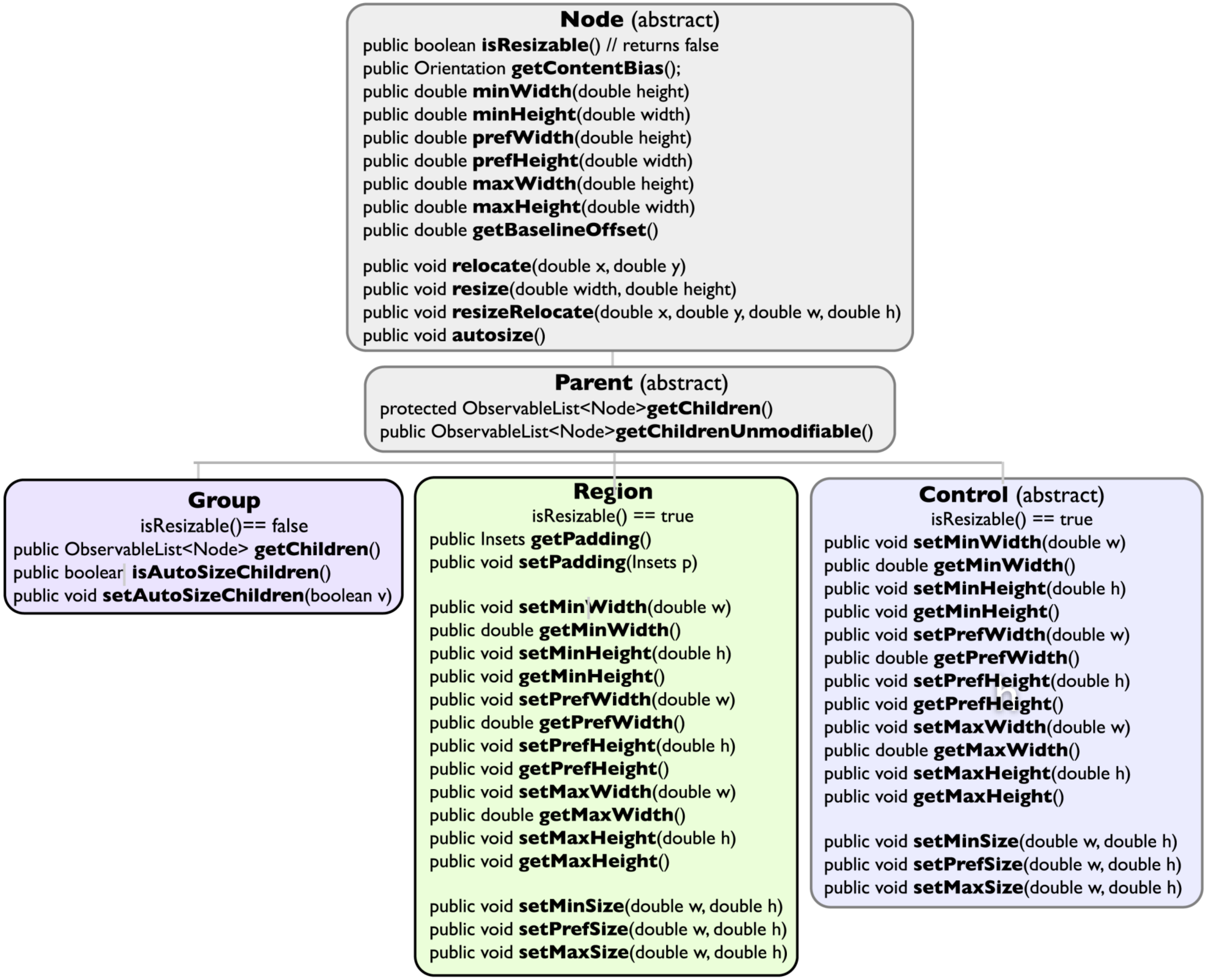


JavaFX 2.0 Layout Classes



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 boolear **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 void **getMinHeight**()
public void **setPrefWidth**(double w)
public double **getPrefWidth**()
public void **setPrefHeight**(double h)
public void **getPrefHeight**()
public void **setMaxWidth**(double w)
public double **getMaxWidth**()
public void **setMaxHeight**(double h)
public void **getMaxHeight**()

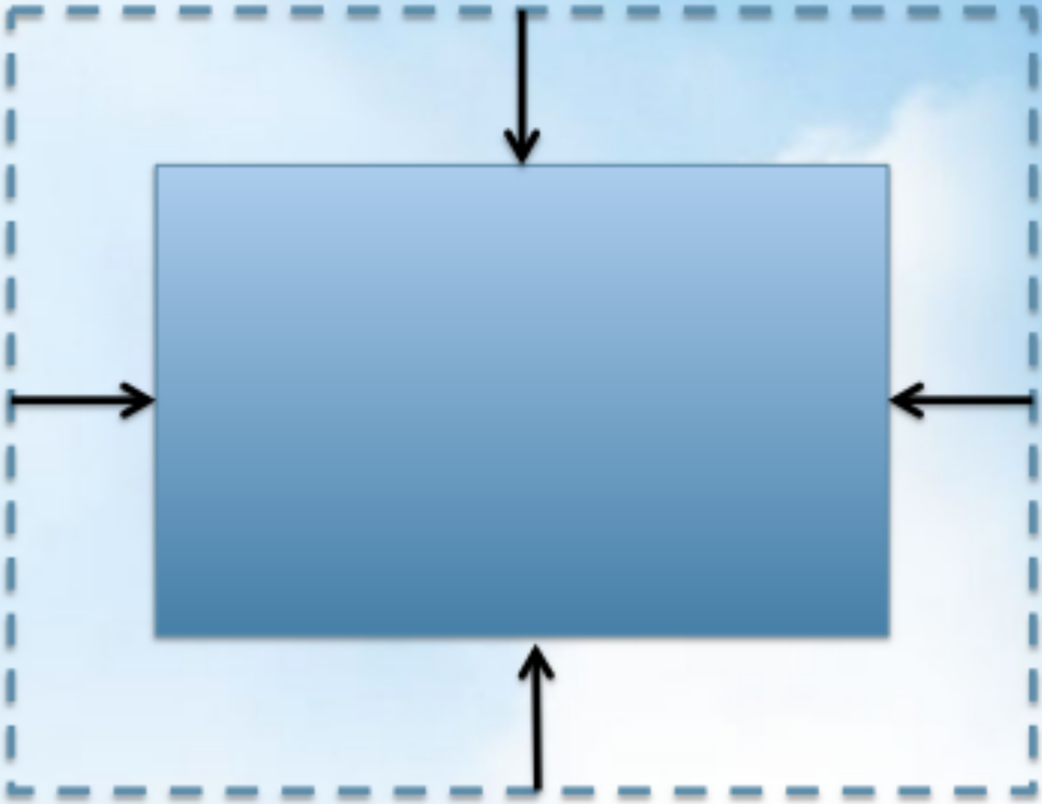
public void **setMinSize**(double w, double h)
public void **setPrefSize**(double w, double h)
public void **setMaxSize**(double w, double h)

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

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

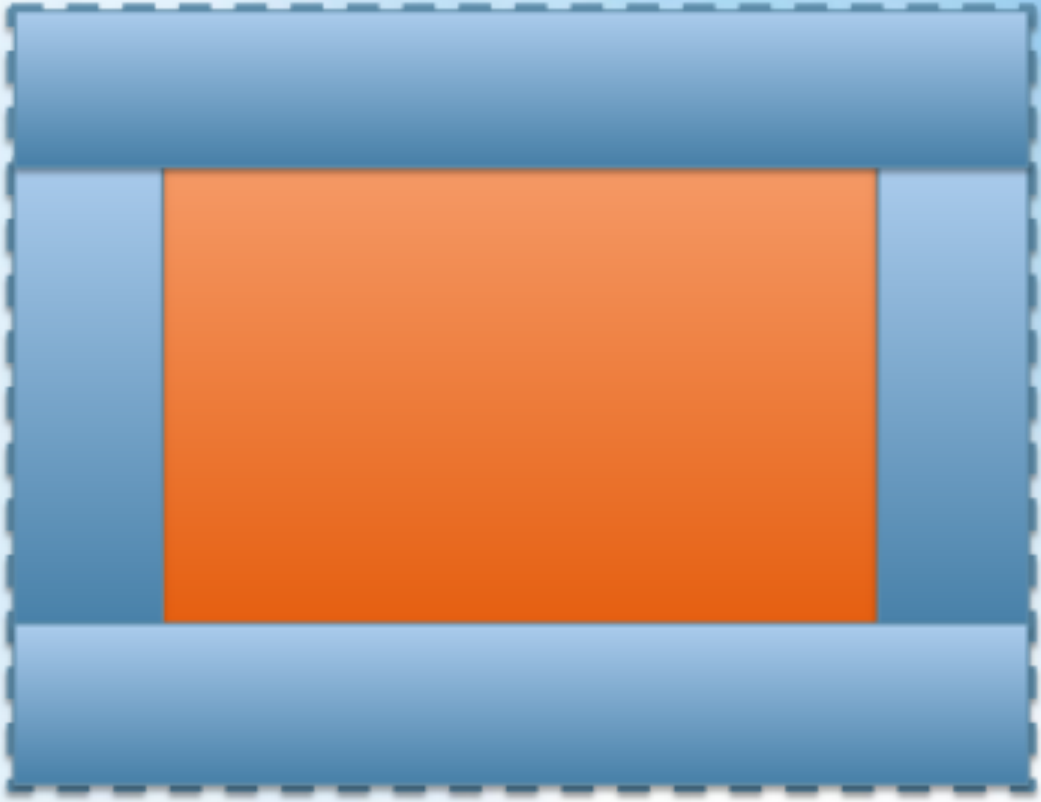
Built-in Layouts

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



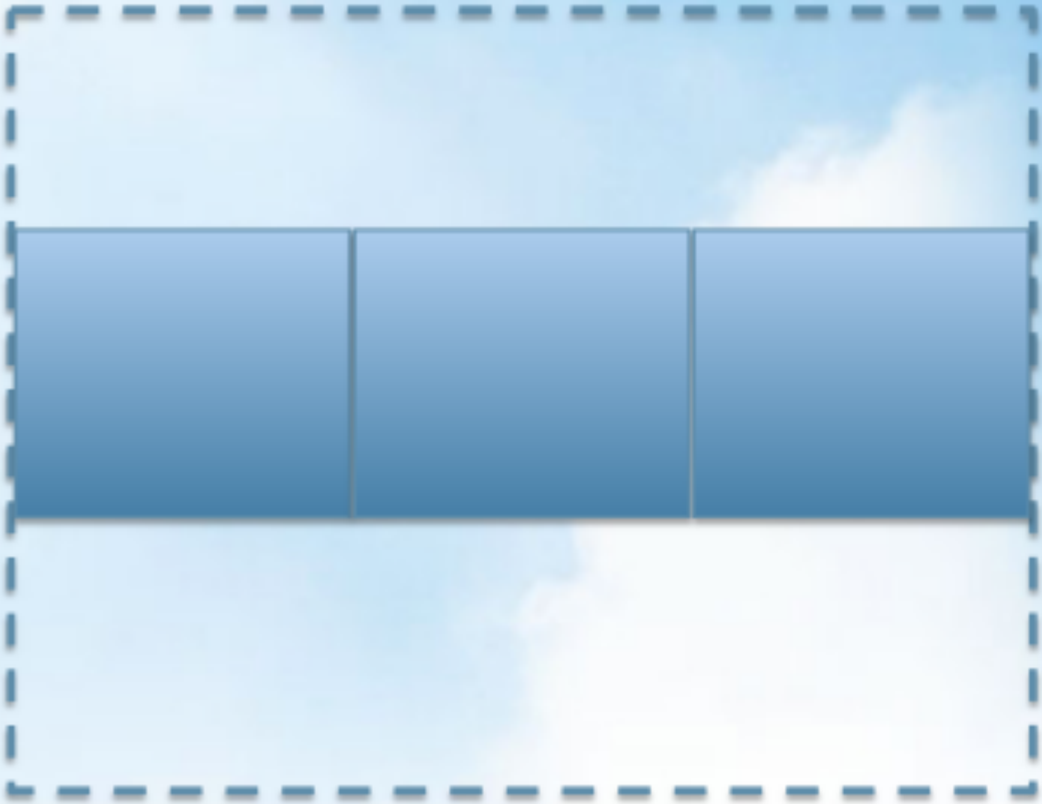
Built-in Layouts

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



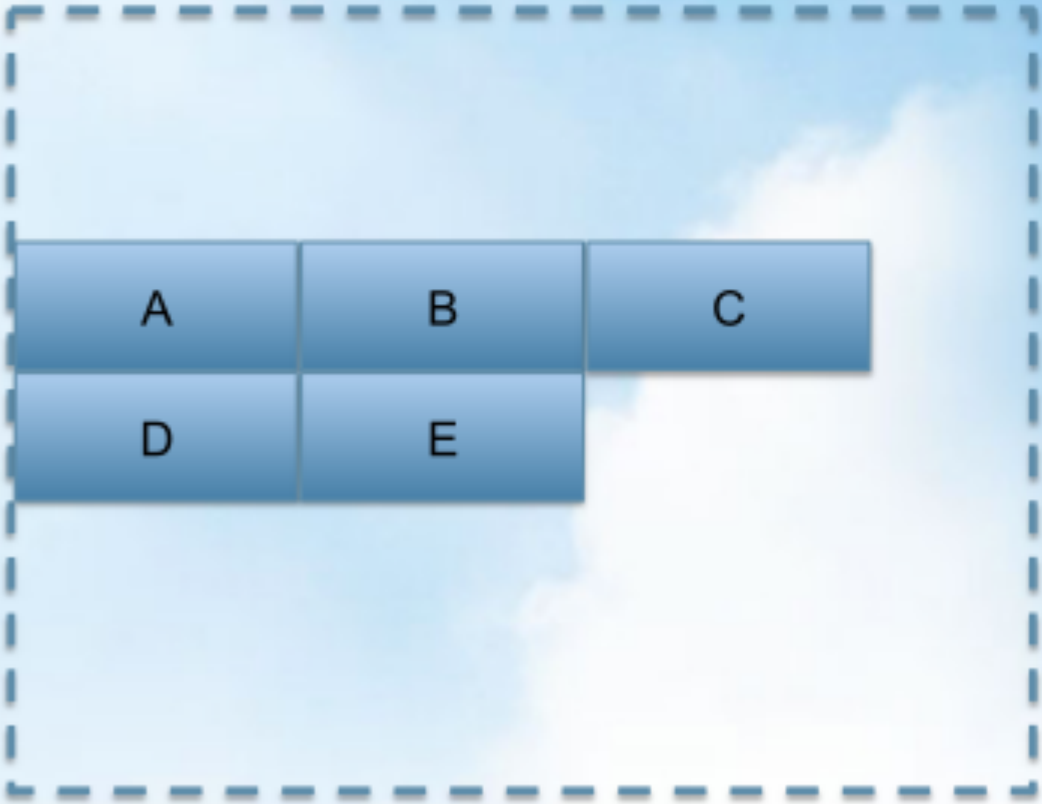
Built-in Layouts

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



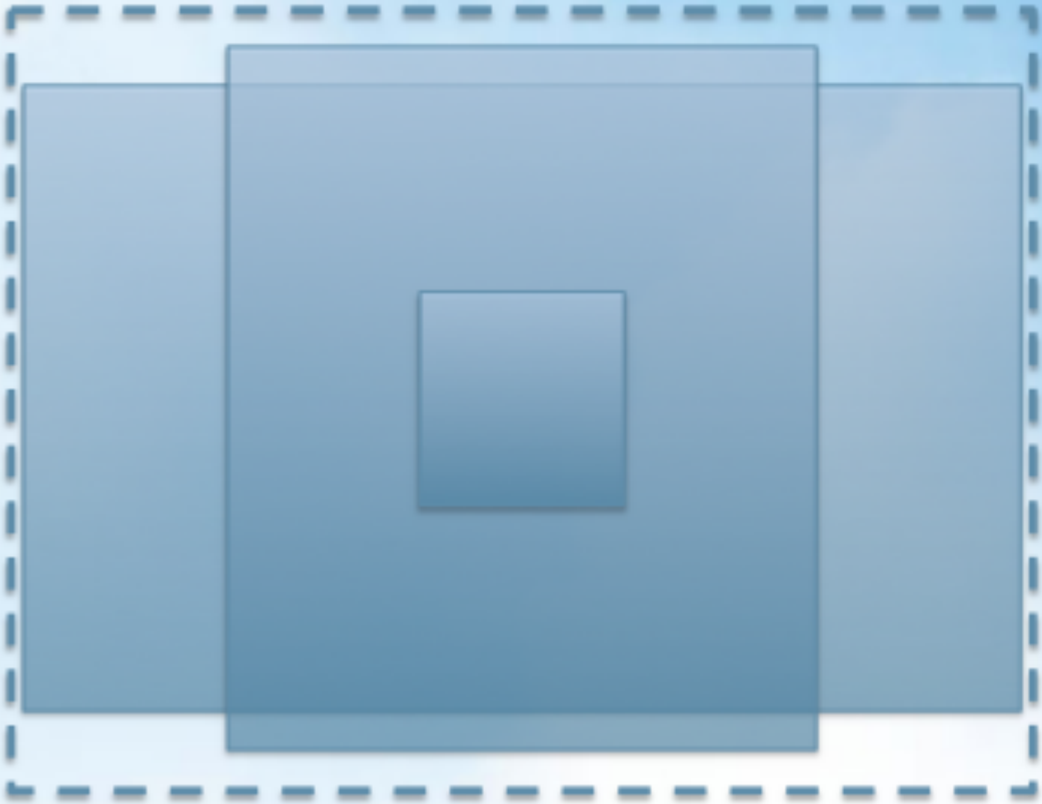
Built-in Layouts

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



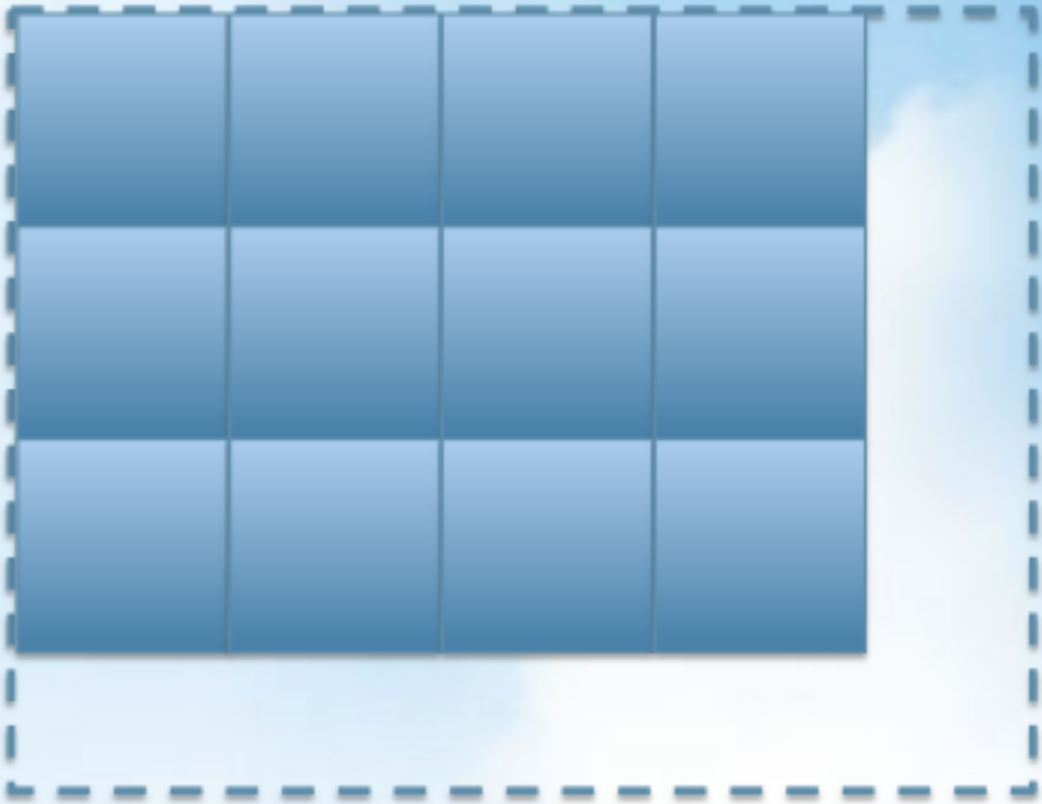
Built-in Layouts

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



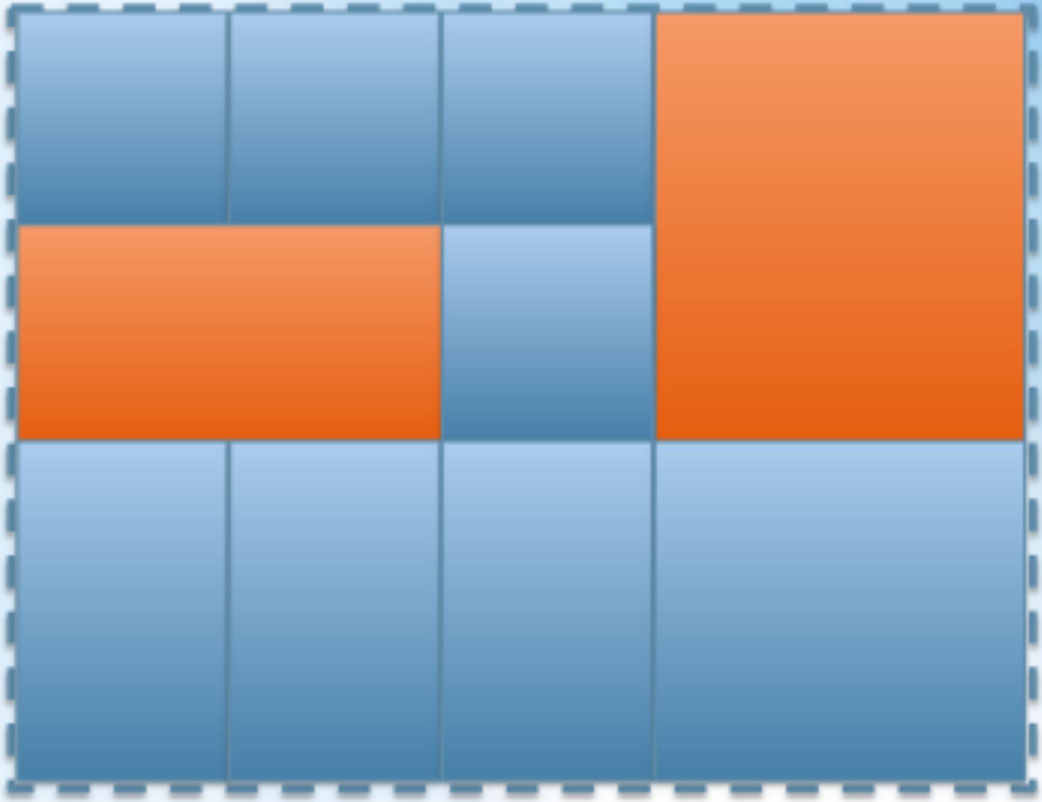
Built-in Layouts

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



Built-in Layouts

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



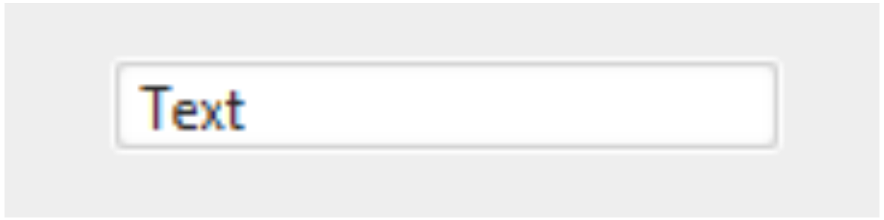
Creating the Scene Graph

- 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() ;

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

Builders

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

        primaryStage.show();
}
```

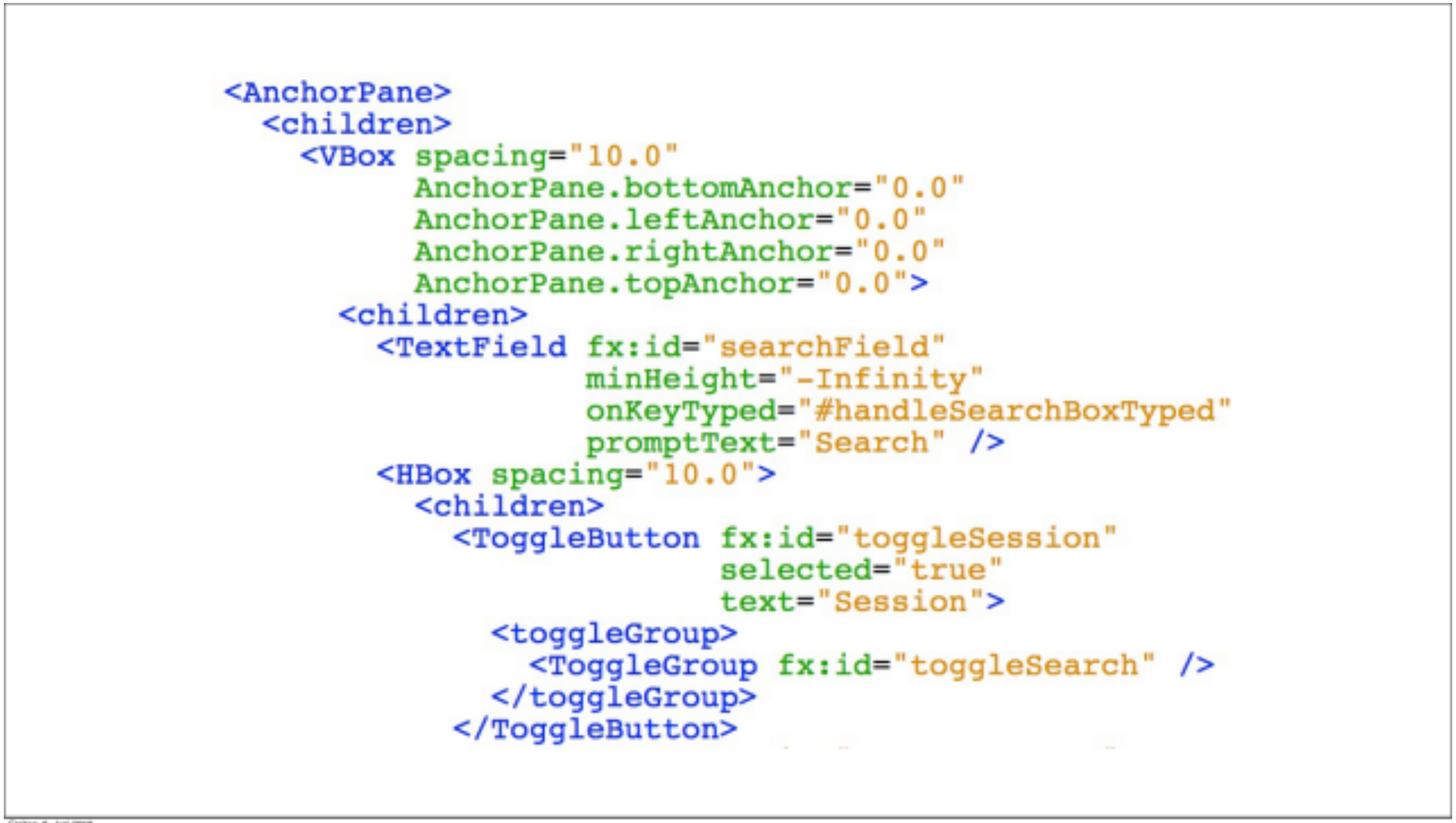
The FXML way…

- 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

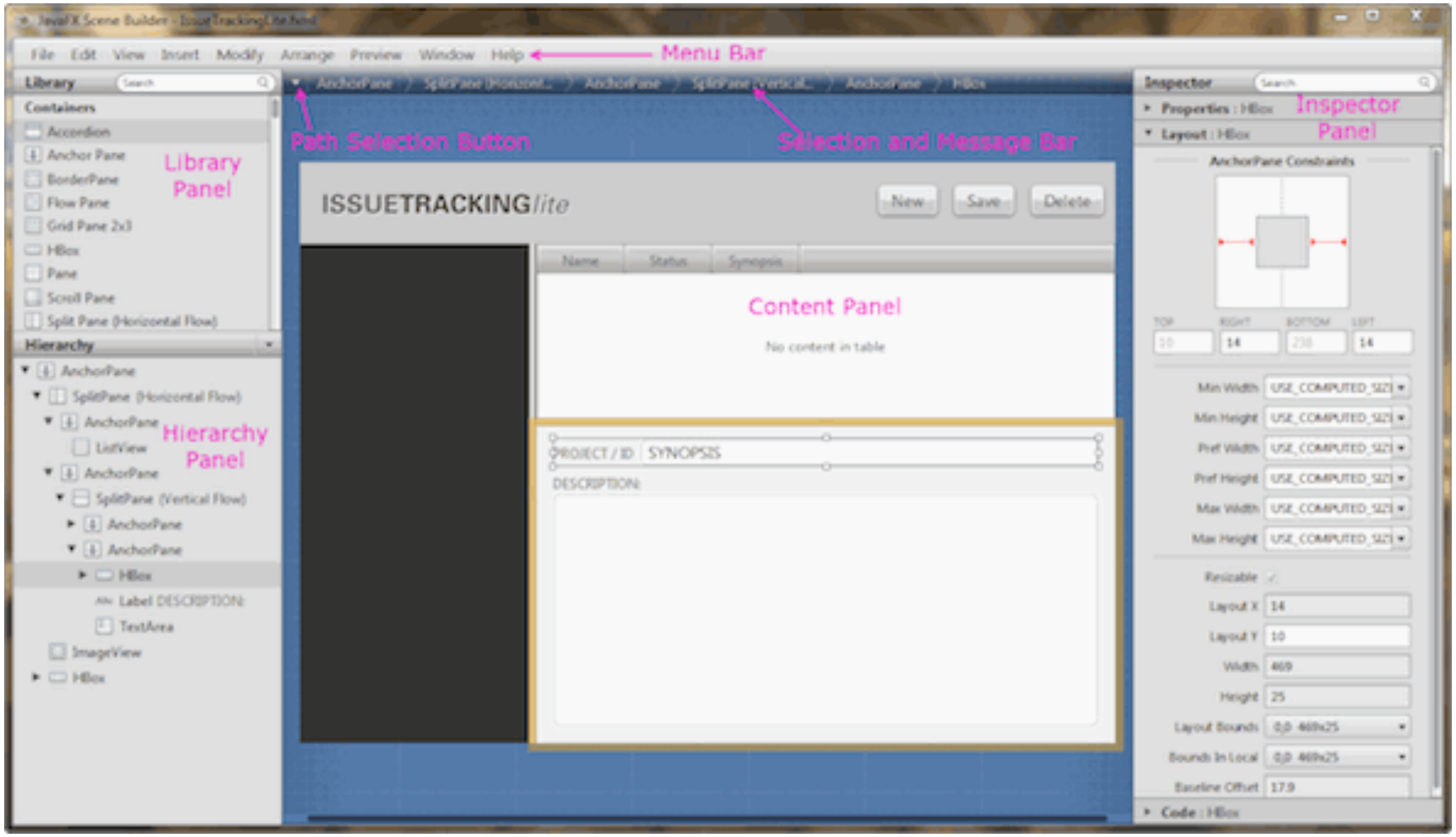
Example



```
<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>
```

Freitag, 8. Juni 2012

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)

Events

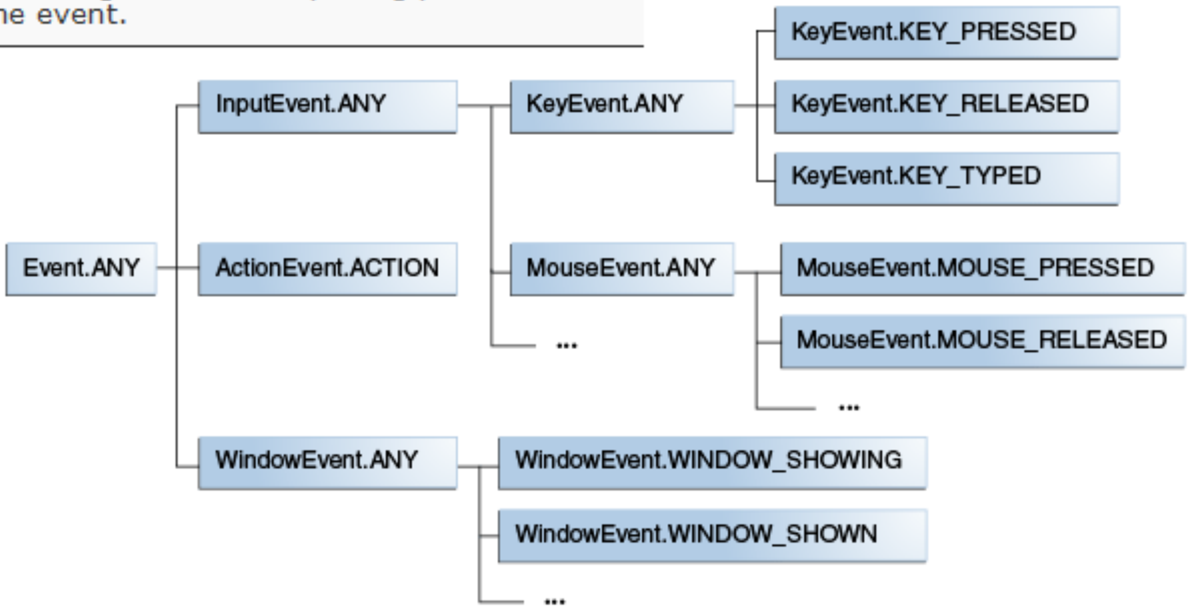
Introduction to JavaFX

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.

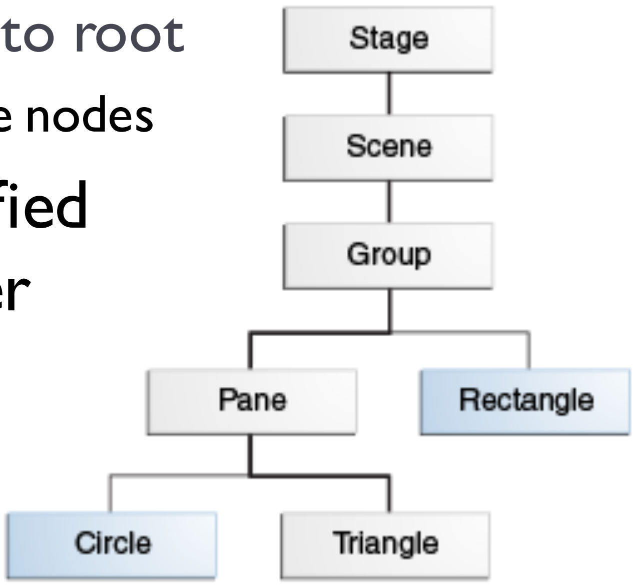
What is an event?

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

- `setOn`*`Event-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");
            }

        });
```

Model-View-Controller

JavaFX programming

Application complexity and MVC

- Interactive, graphical applications exhibit complex interaction patterns
- Flow of control is in the hand of the user
- Actions are mainly asynchronous

- How to organize the program?
- Where to store data?
- How to decouple application logic from interface details?
- How to keep in sync the inner data with the visibile interface?

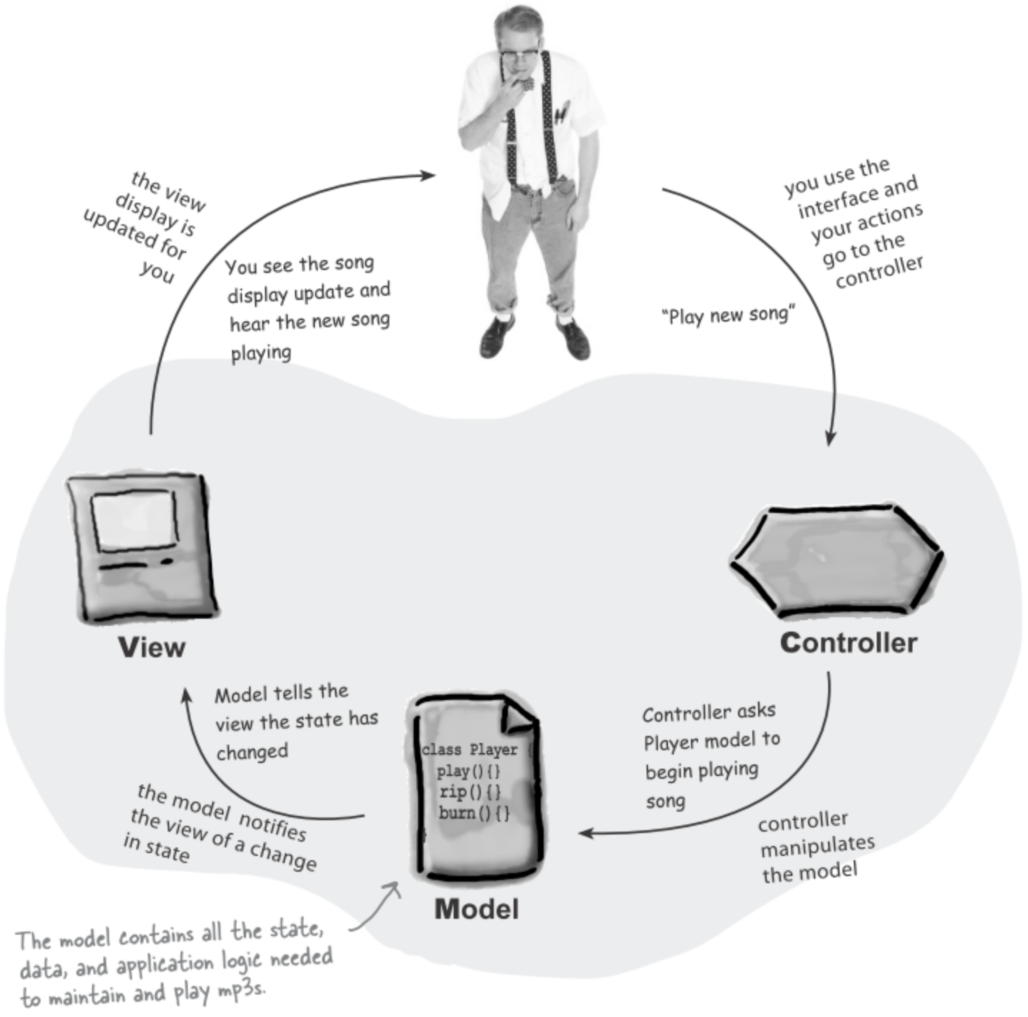
Design Patterns

O'REILLY
10th Anniversary
Updated for Java 8
Head First
Design Patterns
A Brain-Friendly Guide
Avoid those embarrassing coupling mistakes
Learn why everything your friends know about Factory pattern is probably wrong
Discover the secrets of the Patterns Guru
Load the patterns that matter straight into your brain
Find out how Starbuzz Coffee doubled their stock price with the Decorator pattern
See why Jim's love life improved when he cut down his inheritance
Eric Freeman & Elisabeth Robson
with Kathy Sierra & Bert Bates

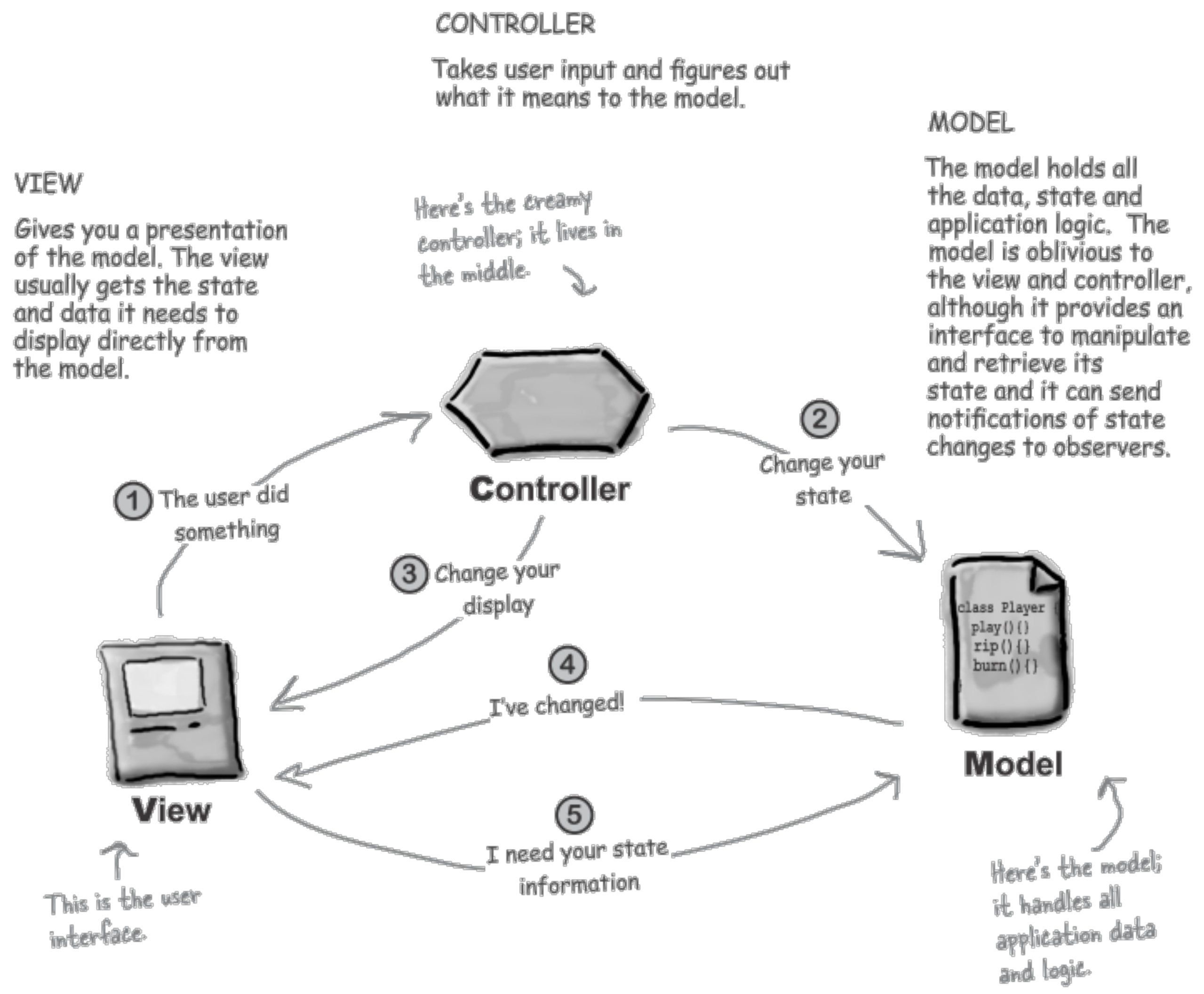
Design Patterns

- How to build systems with good OO design qualities
  - Reusable, extensible, maintainable
- Patterns: Proven solutions to recurrent problems
  - Design problems
  - Programming problems
- Adopt and combine the OO constructs
  - Interface, inheritance, abstract classes, information hiding, polymorphism, objects, statics, …
- Help dealing with *changes* in software
  - Some part of a system is free to vary, independently from the rest

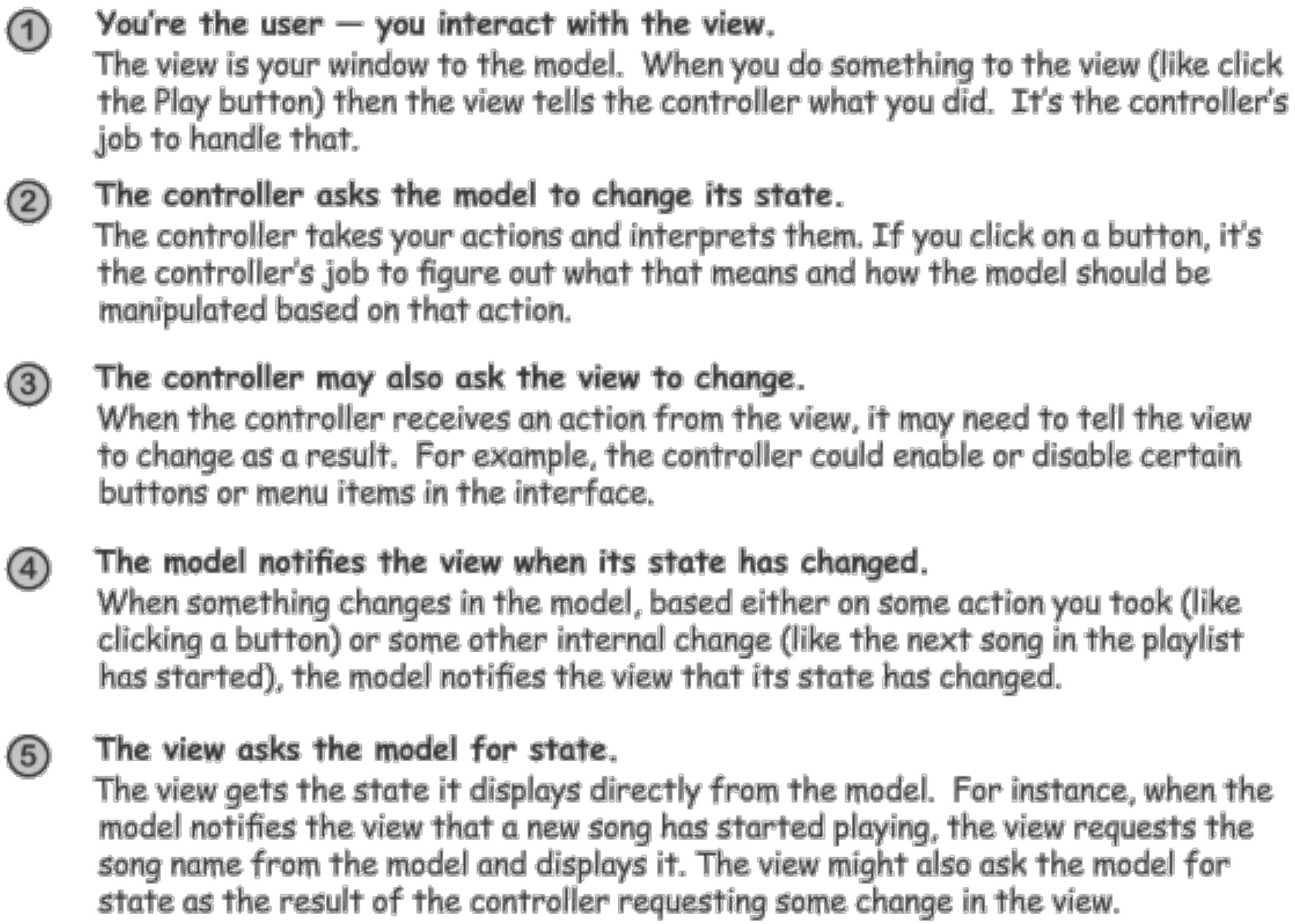
Media Player example


the view display is updated for you
You see the song display update and hear the new song playing
you use the interface and your actions go to the controller
"Play new song"
View
Controller
Model tells the view the state has changed
the model notifies the view of a change in state
class Player {
play() {}
rip() {}
burn() {}
Model
Controller asks Player model to begin playing song
controller manipulates the model
The model contains all the state, data, and application logic needed to maintain and play mp3s.

MVC pattern defined


CONTROLLER
Takes user input and figures out what it means to the model.
MODEL
The model holds all the data, state and application logic. The model is oblivious to the view and controller, although it provides an interface to manipulate and retrieve its state and it can send notifications of state changes to observers.
VIEW
Gives you a presentation of the model. The view usually gets the state and data it needs to display directly from the model.
Here's the creamy controller; it lives in the middle.
Controller
1 The user did something
2 Change your state
3 Change your display
4 I've changed!
5 I need your state information
class Player
play(){}
rip(){}
burn(){}
Model
View
This is the user interface.
Here's the model; it handles all application data and logic.

Normal life-cycle of interaction



1. **You're the user — you interact with the view.**
   The view is your window to the model. When you do something to the view (like click the Play button) then the view tells the controller what you did. It's the controller's job to handle that.

2. **The controller asks the model to change its state.**
   The controller takes your actions and interprets them. If you click on a button, it's the controller's job to figure out what that means and how the model should be manipulated based on that action.

3. **The controller may also ask the view to change.**
   When the controller receives an action from the view, it may need to tell the view to change as a result. For example, the controller could enable or disable certain buttons or menu items in the interface.

4. **The model notifies the view when its state has changed.**
   When something changes in the model, based either on some action you took (like clicking a button) or some other internal change (like the next song in the playlist has started), the model notifies the view that its state has changed.

5. **The view asks the model for state.**
   The view gets the state it displays directly from the model. For instance, when the model notifies the view that a new song has started playing, the view requests the song name from the model and displays it. The view might also ask the model for state as the result of the controller requesting some change in the view.

Mapping concepts to JavaFX

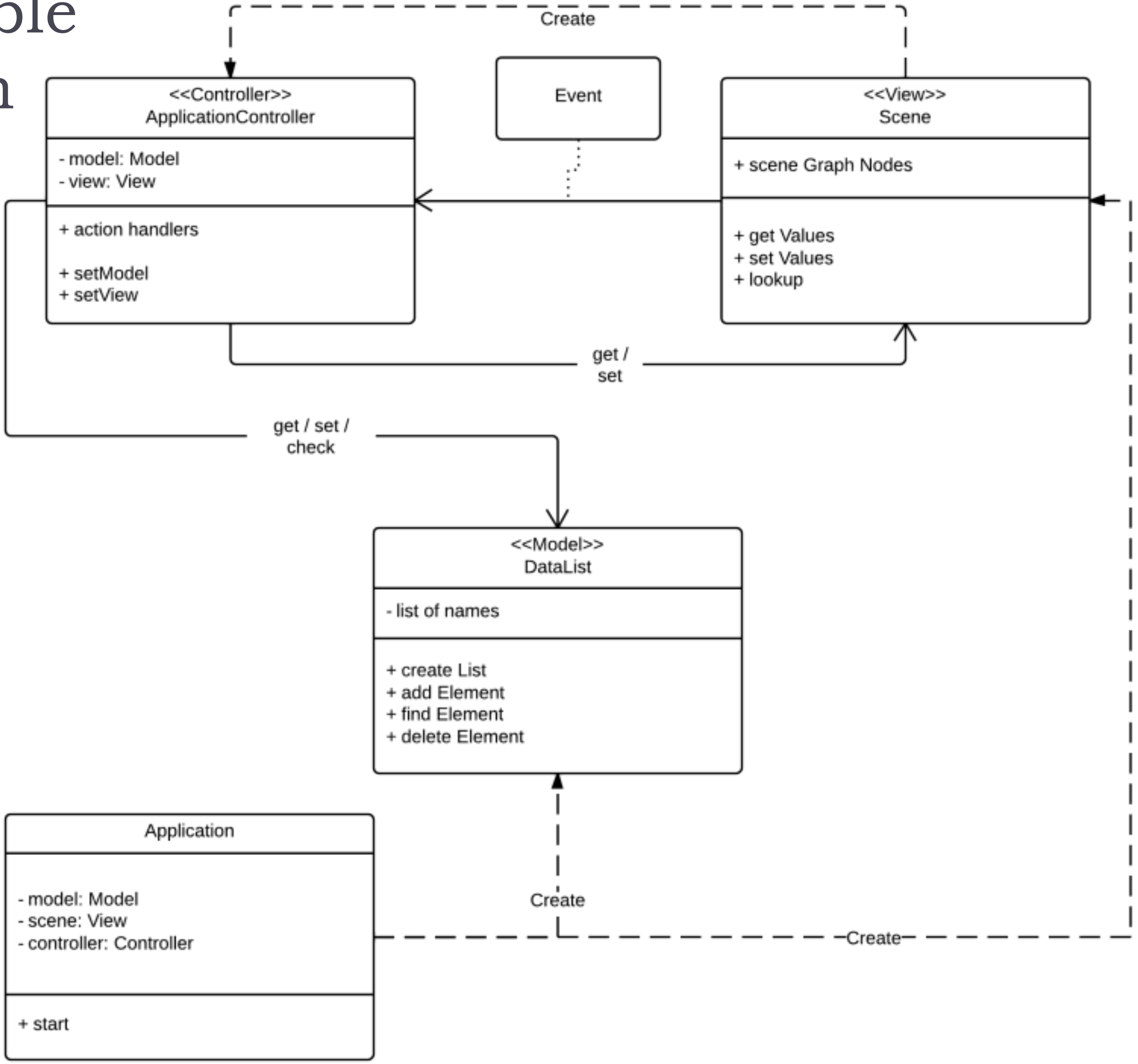
- View: presenting the UI
  - FXML
  - The Nodes in the Scene Graph
- Controller: reacting to user actions
  - Set of event handlers
- Model: handling the data
  - Class(es) including data
  - Persistent data in Data Bases

Design Exercise

- Imagine an application managing a list of items (e.g., names)
- Different items in the user interface should manage the same set of data, with different criteria and actions

- Where do you declare the data class?
- Which class should have access to which?
- Who creates what objects?

A possible solution



The Controller in FXML

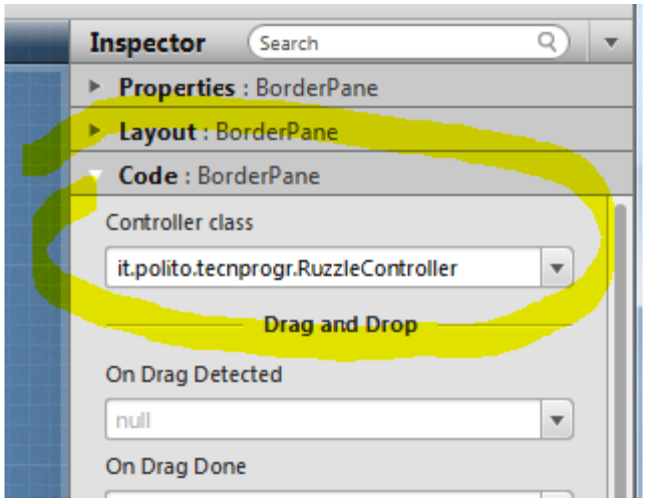
JavaFX programming

The Controller in FXML

- Several attributes in FXML help in the definition of the Controller behavior associated to a scene
  - Identification of the Controller class
  - Injection of Node identifiers (references)
  - Registration of event handlers
- Additionally, the JavaFX Scene Builder may generate a «controller skeleton» for inclusion in the project

Defining the Controller class

- The Root element of the scene graph may specify a **`fx:controller`** attribute
  - `<BorderPane id="BorderPane" xmlns:fx="http://javafx.com/fxml"` **`fx:controller="it.polito.tecnprogr.RuzzleController"`**`>`


Inspector
Properties : BorderPane
Layout : BorderPane
Code : BorderPane
Controller class
it.polito.tecnprogr.RuzzleController
Drag and Drop
On Drag Detected
null
On Drag Done

fx:controller attribute

- Associate a "controller" class with an FXML document
  - Automatically create the instance when FXML is loaded
- Should include event handler methods
- May include an initialize() method
  - public void initialize();
  - called once when the contents of its associated document have been completely loaded
  - any necessary post-processing on the content

Accessing the controller instance

- The Application often needs to communicate with the controller object
  - E.g., to call setModel()
- FXMLLoader provides this information

```
URL location = getClass().getResource("example.fxml");

FXMLLoader fxmlLoader = new FXMLLoader(location);

Pane root = (Pane)fxmlLoader.load();

MyController controller =
(MyController)fxmlLoader.getController();
```

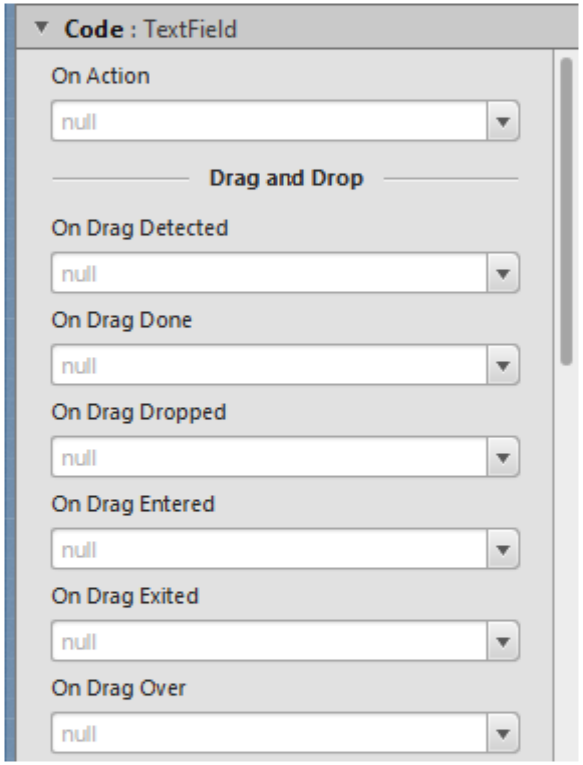
Injection of Node references

- The controller code may directly access various Nodes in the associated scene graph
- The attribute @FXML associates a Node variable with the corresponding node, with the same fx:id value as the variable name
  - No more error-prone «lookup» calls...
  - Local variables in the controller instance
- Try: View | Show Sample Controller Skeleton on the Scene Builder!

```
@FXML // fx:id="theTitle"
   private Label theTitle;
```

Registration of Event Handlers

- In FXML, you may set a event handler through attributes
  - onAction, onKeyTyped, onMouseClicked, ... hundreds more ...
- The value should be the #name of a method in the controller class
  - With the right signature for the event type


Code : TextField

On Action
null

Drag and Drop

On Drag Detected
null

On Drag Done
null

On Drag Dropped
null

On Drag Entered
null

On Drag Exited
null

On Drag Over
null

```
<Button fx:id="cercaBtn"
onAction="#doCercaParola"
text="Cerca" />
```

```
@FXML
void doCercaParola (
ActionEvent event ) {
```

Properties and bindings

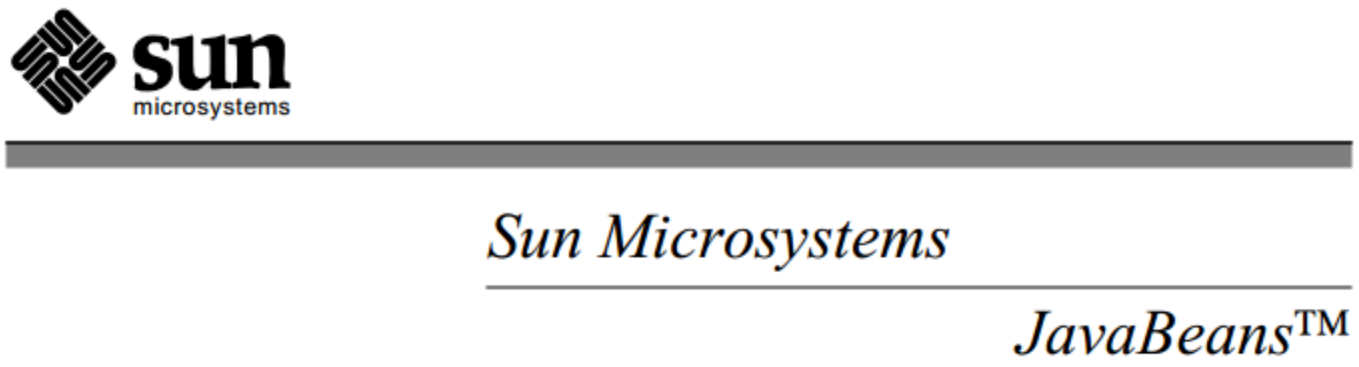
JavaFX programming

JavaFX Properties

- Modern revisitation of the JavaBeans framework (back from Java 1.1)
- Easy to connect different variable values
  - Some may be internal variables
  - Some may be visual elements
- Supports automatic «binding» to efficiently propagate changes
- Simplifies programming

«Old» JavaBeans conventions

- http://www.oracle.com/technetwork/java/javase/documentation/spec-136004.html



sun microsystems

Sun Microsystems

JavaBeans™

Resources

Introduction to JavaFX

Resources

- Official
  - http://www.oracle.com/us/technologies/java/fx/overview/index.html
  - http://www.oracle.com/technetwork/java/javafx/overview/index.html
- Documents
  - http://docs.oracle.com/javafx/
  - **http://docs.oracle.com/javafx/2/api/index.html**
- Blogs
  - http://fxexperience.com/
  - http://www.learnjavafx.typepad.com/weblog/
  - http://community.java.net/community/javafx

Resources

- API
  - http://docs.oracle.com/javafx/2/api/index.html
- Slides/Tips
  - http://www.slideshare.net/steveonjava/java-fx-20-a-developers-guide
  - http://refcardz.dzone.com/refcardz/getting-started-javafx
- Tutorials/Articles
  - http://docs.oracle.com/javafx/2/events/jfxpub-events.htm
  - http://amyfowlersblog.wordpress.com/2011/06/02/javafx2-0-layout-a-class-tour/
- Examples (Downloads)
  - JavaFX Demos and Samples, at http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

Resources

- FXML Controller
  - http://docs.oracle.com/javafx/2/api/javafx/fxml/doc-files/introduction_to_fxml.html#controllers
- Charts
  - Using JavaFX Charts tutorial: http://docs.oracle.com/javafx/2/charts/jfxpub-charts.htm
- Books
  - Head First Design Patterns, chapter 12

Resources

- Properties and Bindings
  - http://docs.oracle.com/javafx/2/binding/jfxpub-binding.htm
  - http://thierrywasyl.wordpress.com/2012/07/29/properties-and-bindings-in-javafx/

Licenza d'uso