

Computer System Design & Application

计算机系统设计与应用A

陶伊达 (TAO Yida)

taoyd@sustech.edu.cn

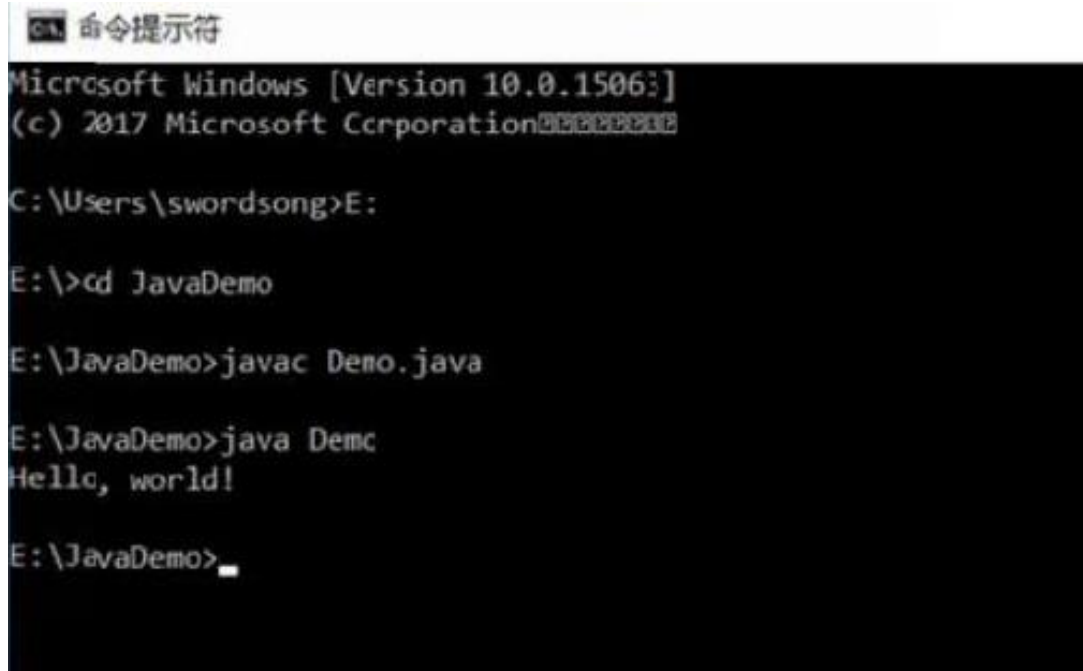
An abstract graphic on the left side of the slide, featuring concentric circles and various digital patterns like squares, rectangles, and lines in shades of blue, green, and white, creating a sense of depth and complexity.

Lecture 9

- Introduction to GUI
- JavaFX

GUI Overview

- **Graphical User Interface (GUI):** a form of user interface that allows users to interact with electronic devices through graphical icons
- Easier to use compared to text-based user interface (e.g., CLI)



```
命令提示符
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation
C:\Users\swordsong>E:

E:\>cd JavaDemo

E:\JavaDemo>javac Demo.java

E:\JavaDemo>java Demc
Hello, world!

E:\JavaDemo>
```



Java GUI History

Abstract Window Toolkit (AWT)

- JDK 1.0
- Most of AWT's UI components have become obsolete

Swing

- JDK 1.2, enhancement of AWT
- Becomes legacy GUI library (only used in old projects)

JavaFX

- JDK 8, replacement to Swing
- Actively maintained and expected to grow in future
- Become a separate module starting from JDK 11

AWT

- **Components:** e.g., Button, Label, and TextField
- **Container:** used to hold components (e.g., Frame, Panel)

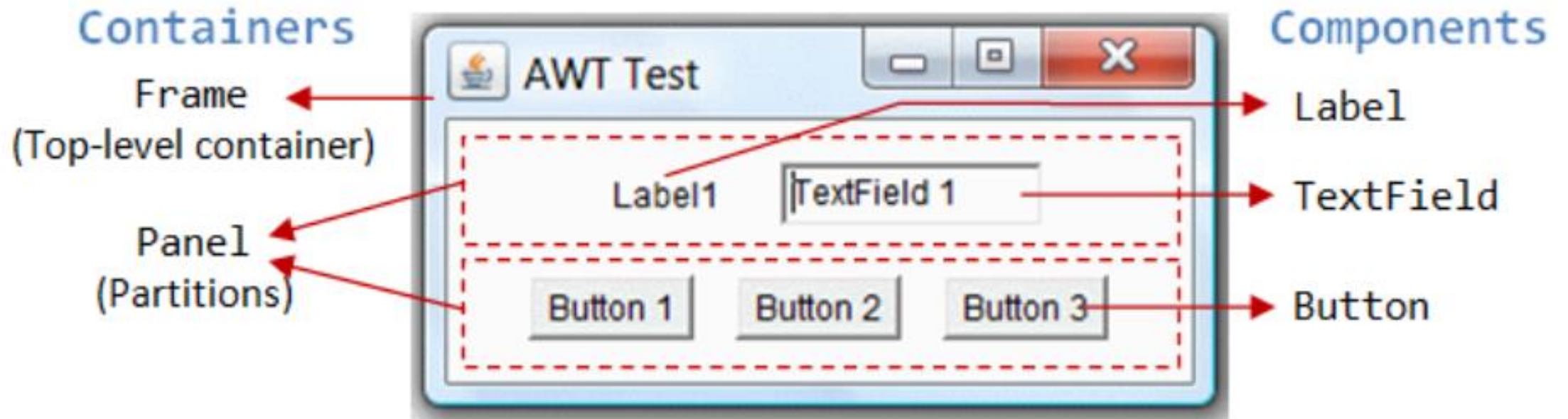


Image source: https://www3.ntu.edu.sg/home/ehchua/programming/java/j4a_gui.html

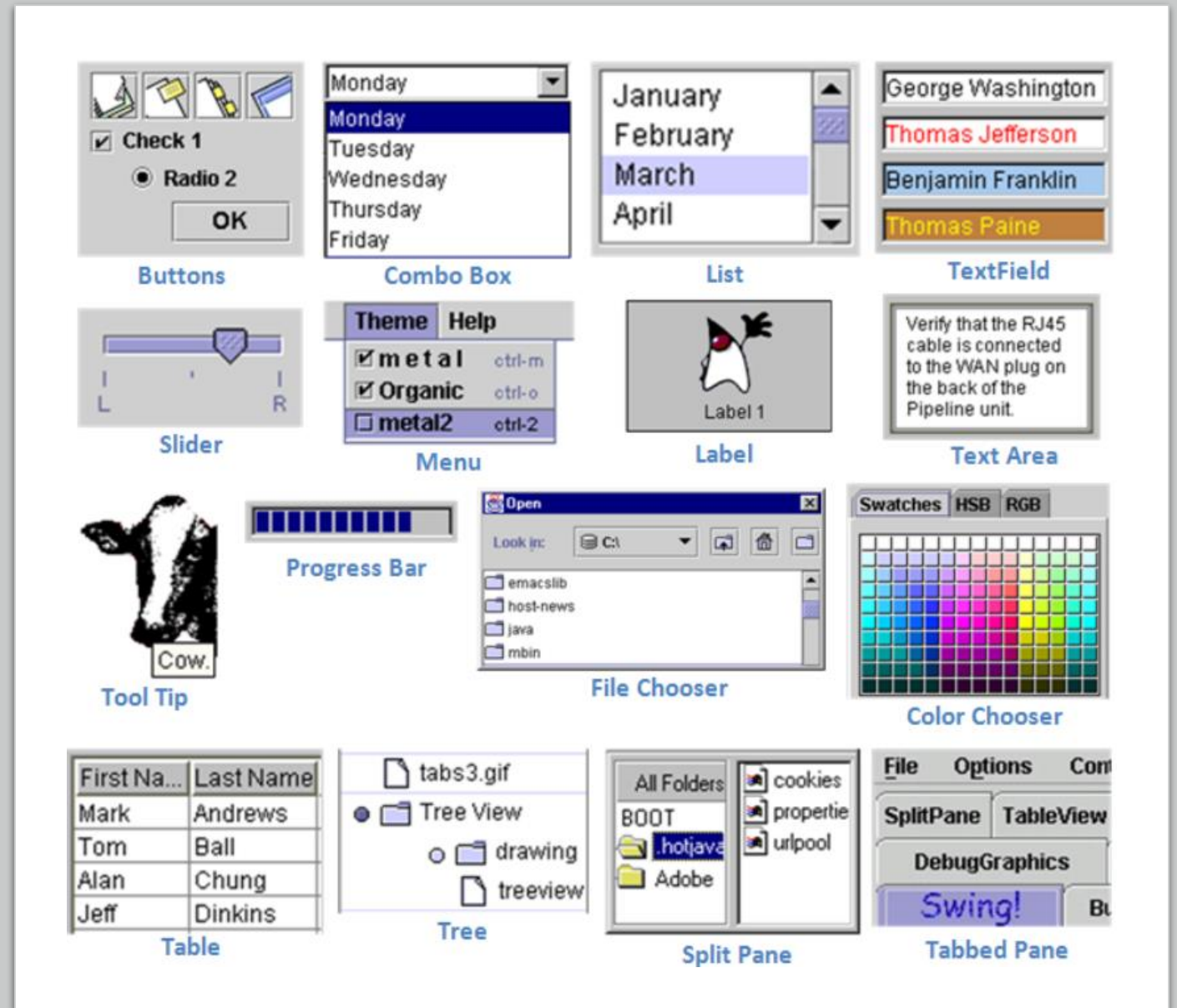
Button Click Event

```
Panel panel = new Panel();  
TextField tf = new TextField();  
Button btn=new Button("click me");  
  
btn.addActionListener(new ActionListener(){  
    public void actionPerformed(ActionEvent e){  
        tf.setText("Welcome");  
    }  
});  
  
panel.add(tf);  
panel.add(btn);
```



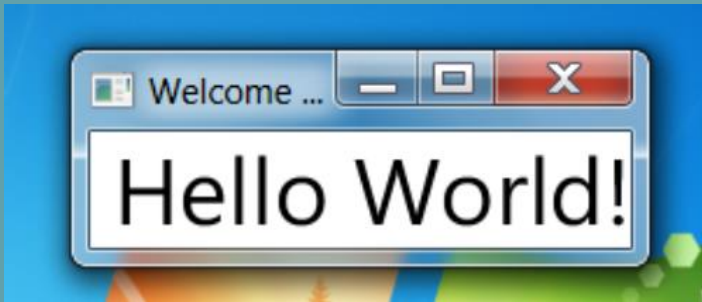
Swing

Swing extends AWT by adding richer graphics functionalities and interactivity to Java applications
(more comprehensive components)



Swing look-and-feel

You can create GUIs that can either look the same across platforms or can assume the look and feel of the current OS platform (such as Microsoft Windows, Linux).



Swing Class Hierarchy

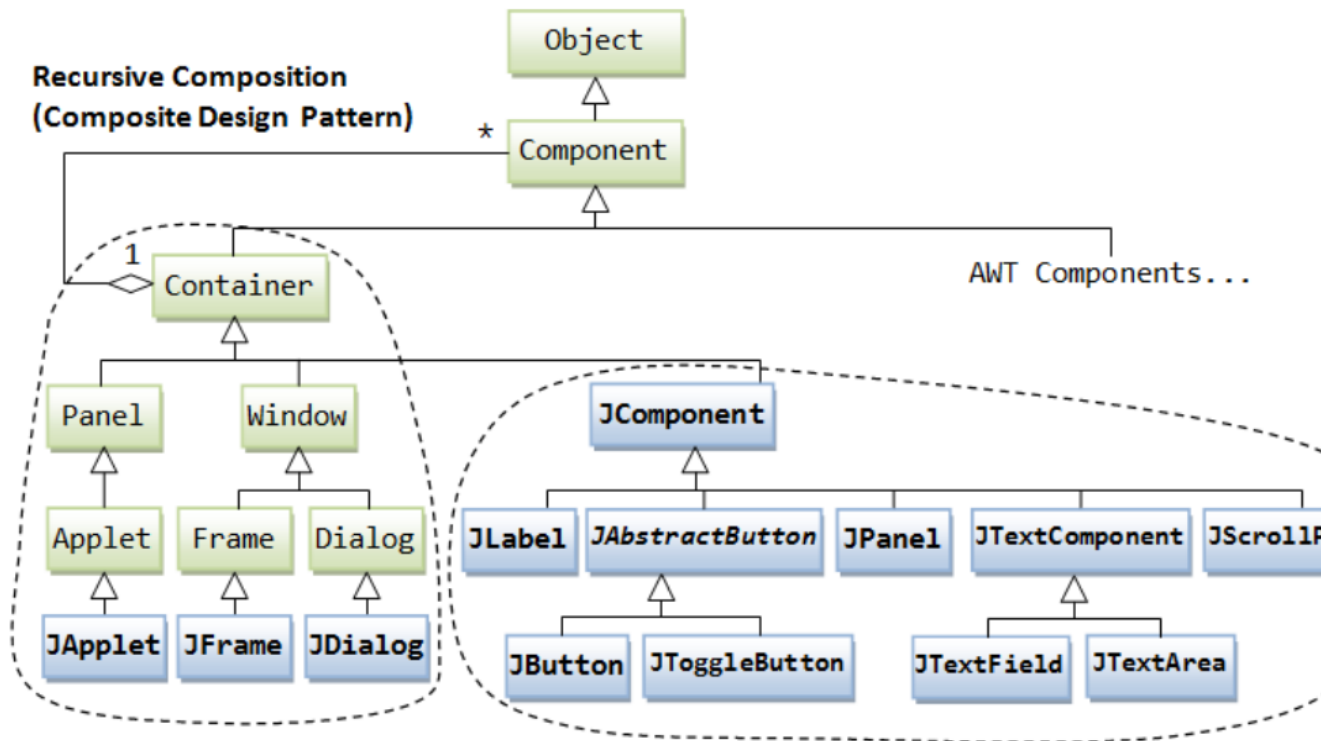


Image source: https://www3.ntu.edu.sg/home/ehchua/programming/java/j4a_gui.html

- Swing also has *containers* and *components*
- Swing component classes (javax.swing) begin with a prefix "J"

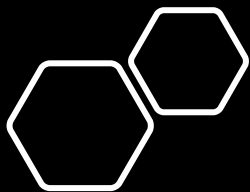
Swing Workflow

1. Make a window (JFrame)
2. Make a container (JPanel)
 - Add it to the window
3. Add components to the container
 - Buttons, textbox, etc
 - Setup layout to control positions
 - Setup listeners to react to events
4. Let the window display the container
5. Wait for the events.....



Lecture 9

- JavaFX
 - Overview
 - Hello World
 - Design & Concepts
 - Layouts, Shapes, UI controls
 - Charts and Axis
 - Transformation, Animation, Effects
 - FXML
 - Multithreading in JavaFX



JavaFX Overview

<https://openjfx.io/>

- Official doc: JavaFX is an open source, next generation client application platform for desktop, mobile and embedded systems built on Java (i.e., a GUI toolkit for Java)
- JavaFX can run on various OS and devices
 - Windows
 - Linux
 - Mac
 - iOS
 - Android/Chromebook
 - Raspberry Pi

JavaFX Showcases

Images from JavaFX official site



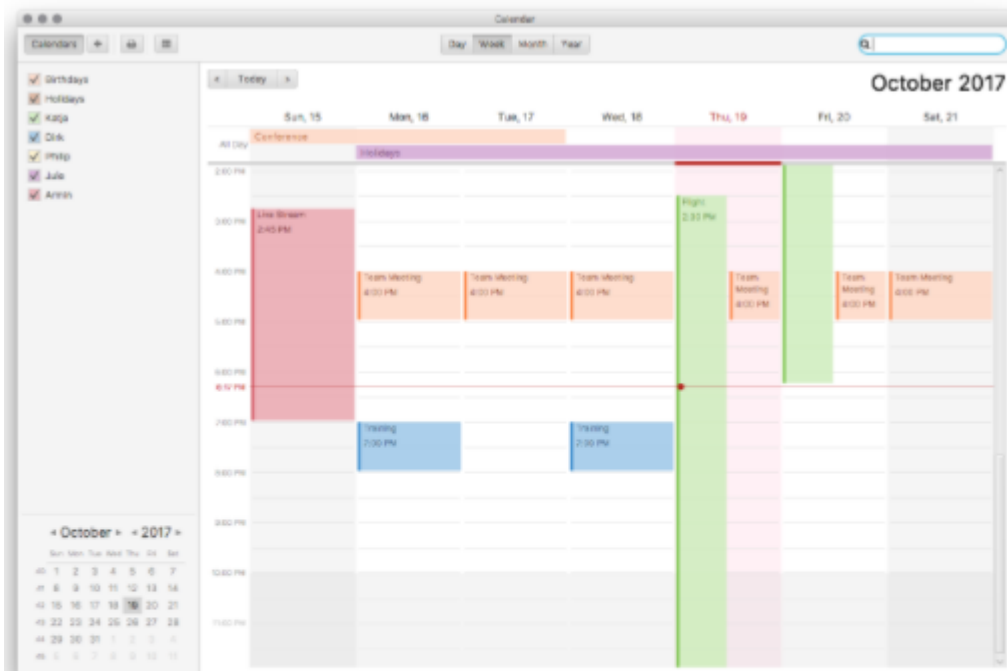
AsciidocFX

An Asciidoc editor to build PDF, Epub, Mobi and HTML books, documents and slides



Gluon Maps

Tiles based geo-location map framework



CalendarFX

A Java framework for creating sophisticated calendar views

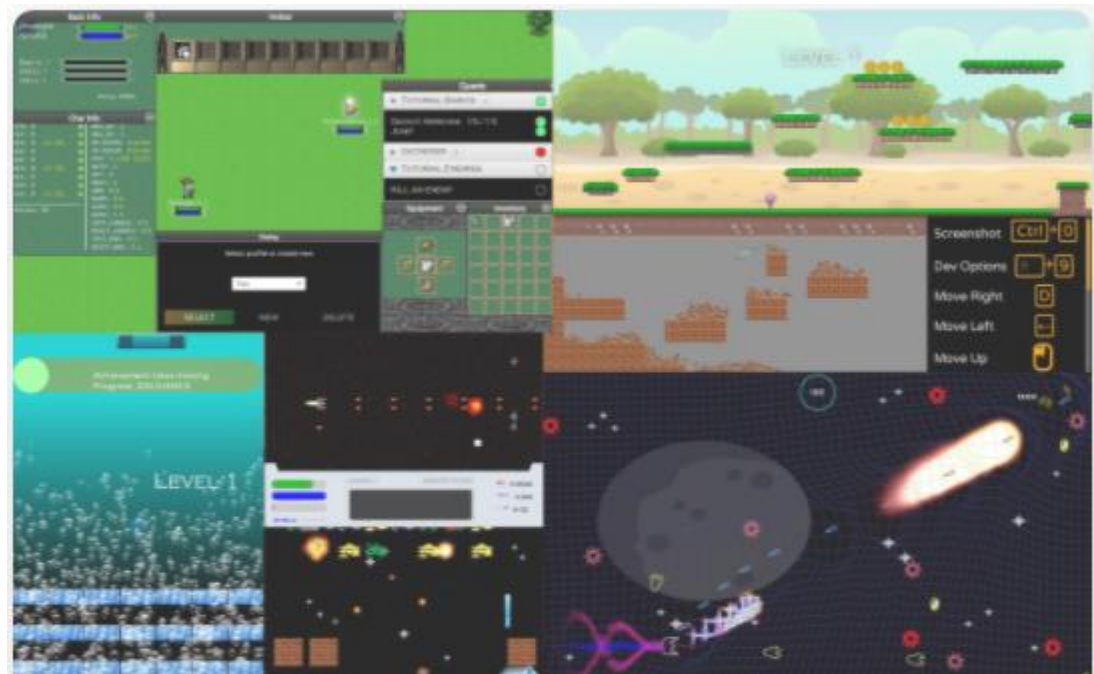
JavaFX Showcases

Images from JavaFX official site



TilesFX

A JavaFX library containing tiles for Dashboards



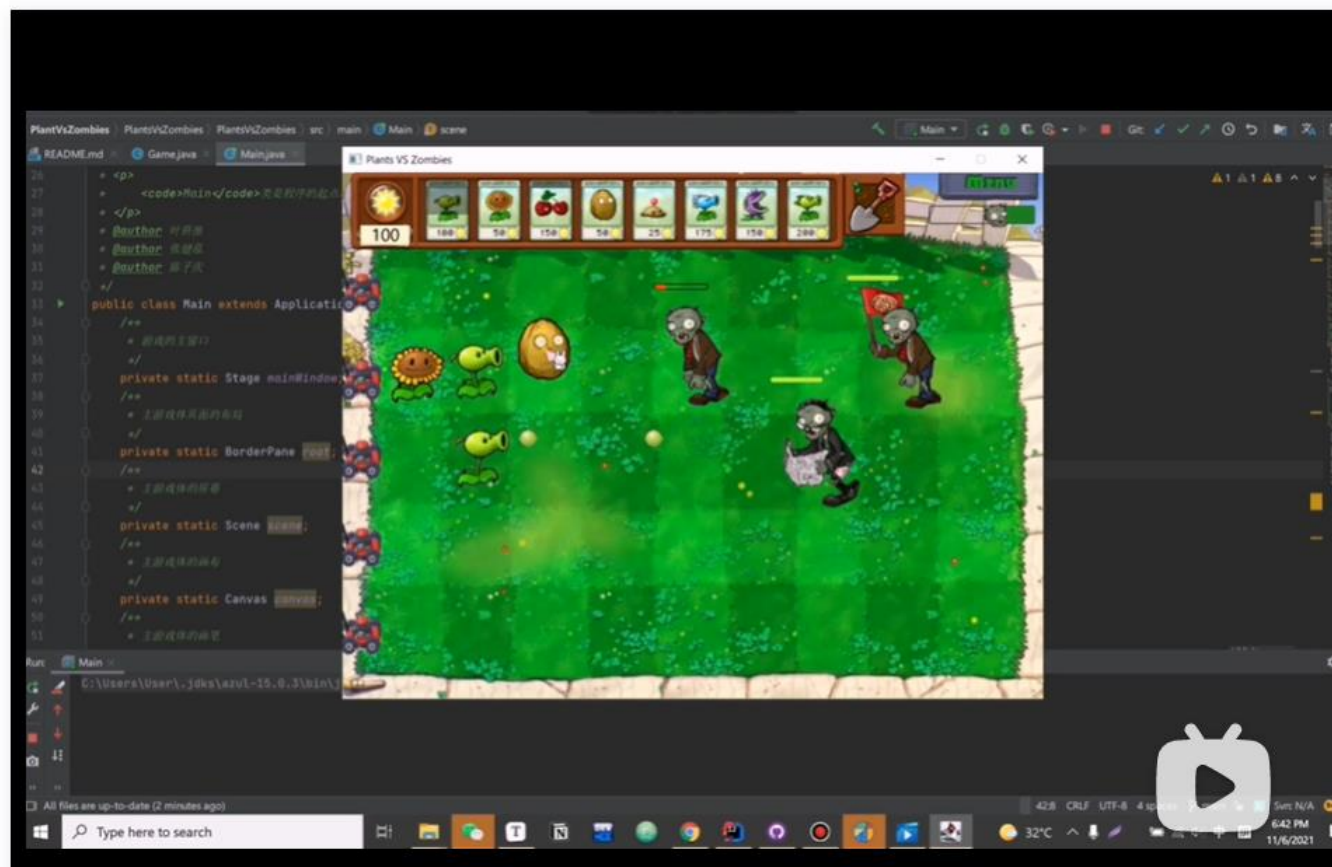
FXGL

JavaFX game engine

JavaFX Showcases

北航1921 C50组大作业 基于JavaFX的植物大战僵尸

8713播放 · 总弹幕数9 2021-06-12 02:10:59





Why do we learn JavaFX?

- A good, real application that applies key OOP principles such as encapsulation, inheritance, and polymorphism
- Learn basic concepts of GUI programming and event-driven programming model, which are applicable to other UI framework and techniques.
- Learn basic design concepts, such as separation of UI and business logic

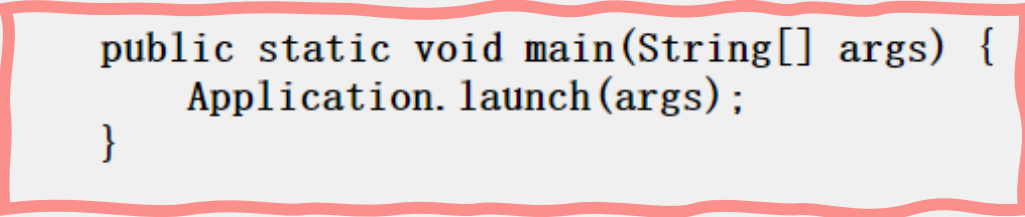
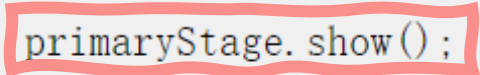
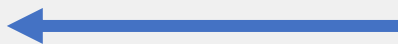
JavaFX Hello World

```
import javafx.application.Application;
import javafx.stage.Stage;

public class MyFxApp extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("My First JavaFX App");
        primaryStage.show();
    }

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



Makes the application visible
(otherwise nothing is shown)

Optional

Import necessary classes from **javafx**

Extend the abstract **Application** class

Implement the abstract **start()** method of the **Application** class (called when a JavaFX application starts)

launch() launches the JavaFX application.

<http://tutorials.jenkov.com/javafx/your-first-javafx-application.html>

JavaFX Hello World

```
import javafx.application.Application;
import javafx.stage.Stage;

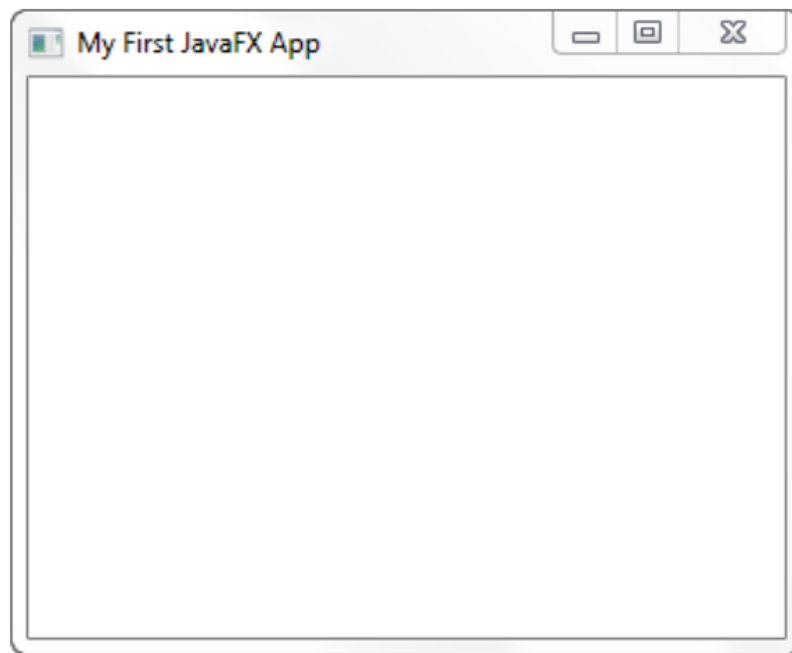
public class MyFxApp extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("My First JavaFX App");

        primaryStage.show();
    }

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

}
```



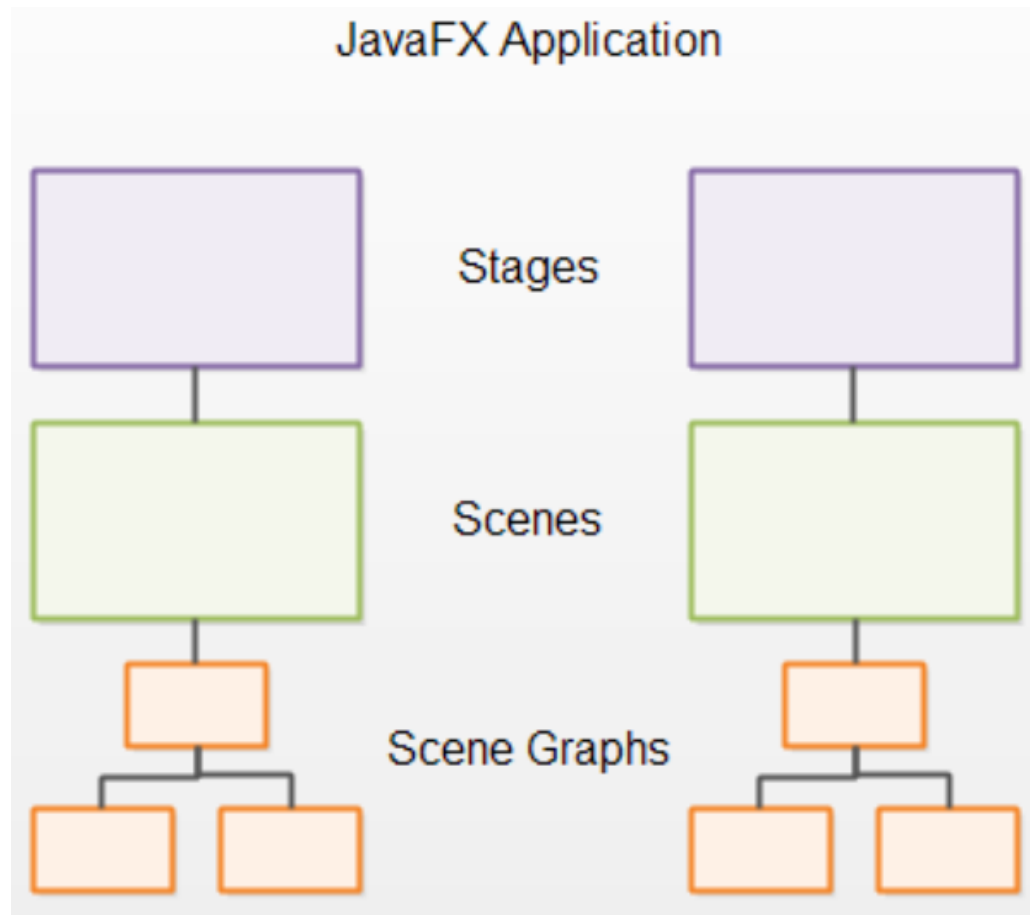
<http://tutorials.jenkov.com/javafx/your-first-javafx-application.html>



Lecture 9

- JavaFX
 - Overview
 - Hello World
 - Design & Concepts
 - Layouts, Shapes, UI controls
 - Charts and Axis
 - Transformation, Animation, Effects
 - FXML
 - Multithreading in JavaFX

JavaFX Design



<http://tutorials.jenkov.com/javafx/your-first-javafx-application.html>

Stage (窗体)

- The outer frame for a JavaFX application, typically corresponds to a window.
- A JavaFX application can have one or more stages (multiple windows open)

Scene (场景)

- Containing all GUI components visible in a window (i.e., to display things on the stage)
- A stage can only show one scene at a time, but it is possible to exchange the scene at runtime

Scene Graphs (场景图)

- All visual components (controls, layouts etc.) attached to a scene is called the scene graph

JavaFX Design

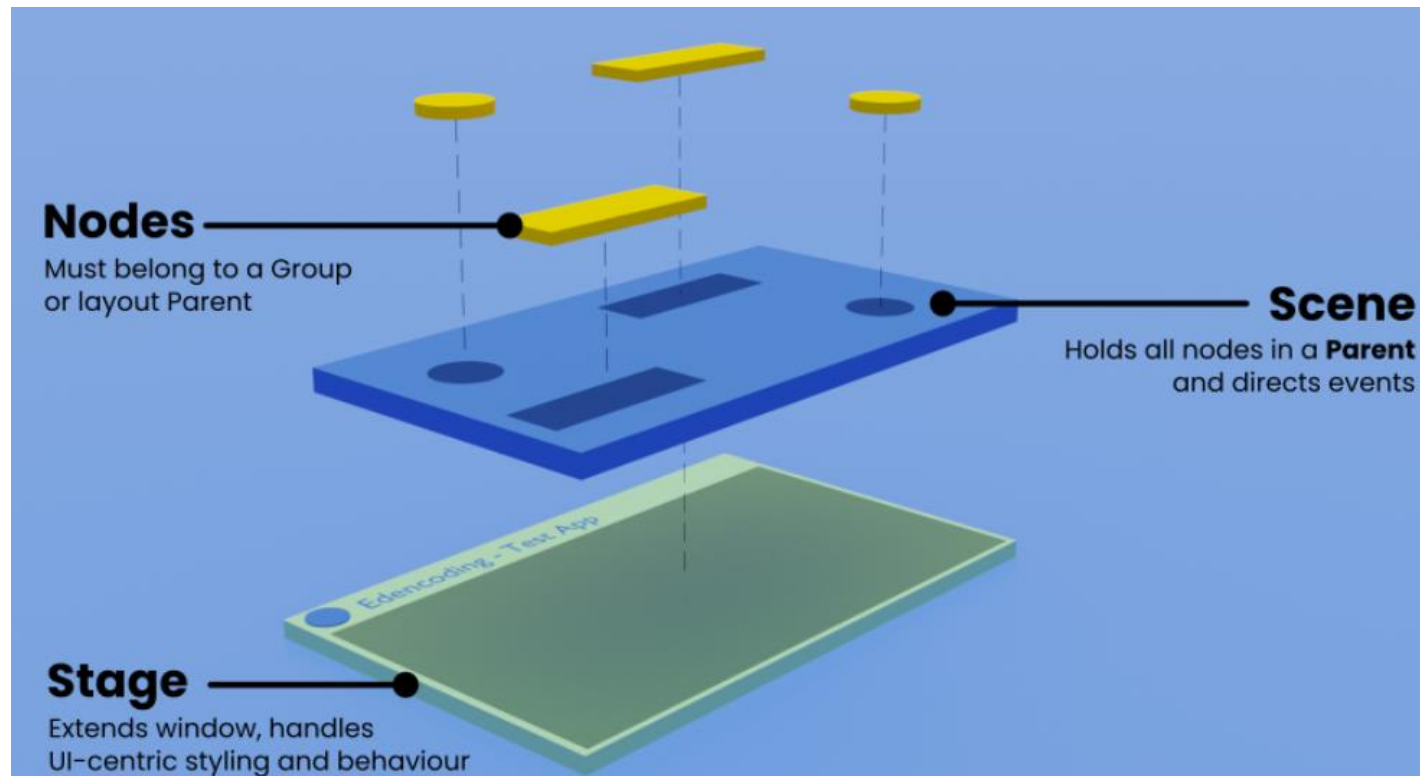


Image source: <https://edencoding.com/javafx-scene/>

JavaFX Stage

- A Stage represents a window in a JavaFX application
- A Stage object is created and passed to the `start(Stage primaryStage)` method when a JavaFX application starts up
- New Stage objects could be created if the application needs to open more windows

```
import javafx.application.Application;
import javafx.stage.Stage;

public class MyFxApp extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("My First JavaFX App");

        primaryStage.show();
    }

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

}
```

New stage could be created by:

```
Stage stage = new Stage();
.....
stage.show();
```

JavaFX Stage Properties

Please refer to the official documentation for full details

<https://docs.oracle.com/javase/8/javafx/api/javafx/stage/Stage.html>

Property and Description

alwaysOnTop

Defines whether this Stage is kept on top of other windows.

fullScreenExitHint

fullScreenExitKey

Get the property for the Full Screen exit key combination.

fullScreen

Specifies whether this Stage should be a full-screen, undecorated.

iconified

Defines whether the Stage is iconified or not.

maxHeight

Defines the maximum height of this Stage.

maximized

Defines whether the Stage is maximized or not.

maxWidth

Defines the maximum width of this Stage.

minHeight

Defines the minimum height of this Stage.

minWidth

Defines the minimum width of this Stage.

resizable

Defines whether the Stage is resizable or not by the user.

title

Defines the title of the Stage.

JavaFX Stage Style

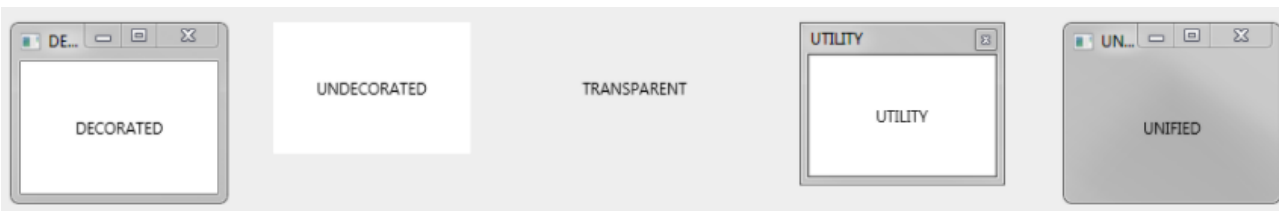
```
stage.initStyle(StageStyle.DECORATED);
```

```
//stage.initStyle(StageStyle.UNDECORATED);
```

```
//stage.initStyle(StageStyle.TRANSPARENT);
```

```
//stage.initStyle(StageStyle.UNIFIED);
```

```
//stage.initStyle(StageStyle.UTILITY);
```



Enum StageStyle

```
java.lang.Object
```

```
java.lang.Enum<StageStyle>
```

```
javafx.stage.StageStyle
```

Enum Constants

Enum Constant and Description

DECORATED

Defines a normal Stage style with a solid white background and platform decorations.

TRANSPARENT

Defines a Stage style with a transparent background and no decorations.

UNDECORATED

Defines a Stage style with a solid white background and no decorations.

UNIFIED

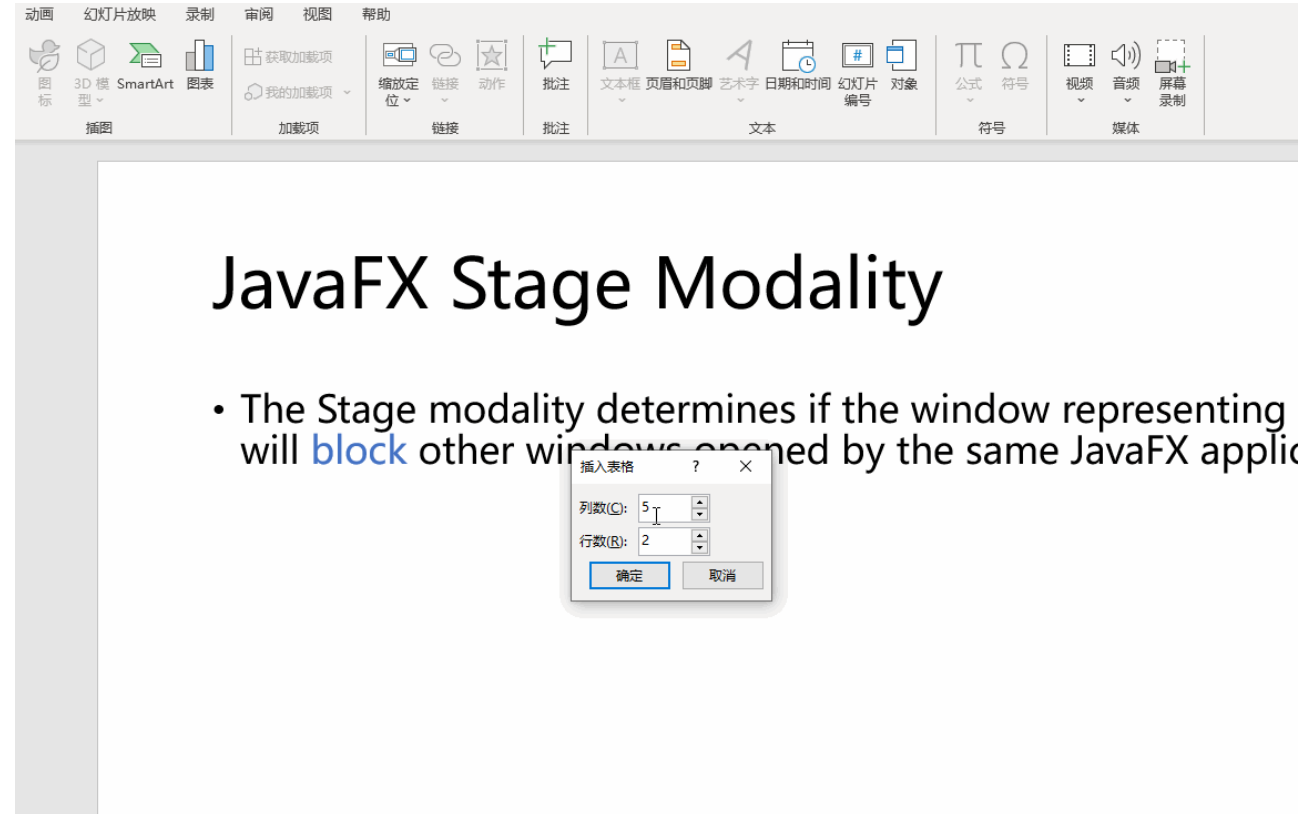
Defines a Stage style with platform decorations and eliminates the border between client area and decorations.

UTILITY

Defines a Stage style with a solid white background and minimal platform decorations used for a utility window.

JavaFX Stage Modality

The Stage modality determines if the window representing the Stage will **block** other windows opened by the same JavaFX application.



The screenshot shows a presentation slide with the title "JavaFX Stage Modality". Below the title is a single bullet point: "The Stage modality determines if the window representing will **block** other windows opened by the same JavaFX applic". A "Insert Table" dialog box is overlaid on the slide, showing "Columns (C): 5" and "Rows (R): 2". The dialog has "确定" (OK) and "取消" (Cancel) buttons. The background of the slide is a light gray with a faint grid pattern.

JavaFX Stage Modality

- The Stage modality determines if the window representing will **block** other windows opened by the same JavaFX applic

Insert Table dialog box:

列数(C):	5
行数(R):	2

确定 取消

JavaFX Stage Modality

Enum Modality

```
java.lang.Object  
    java.lang.Enum<Modality>  
        javafx.stage.Modality
```

Enum Constants

Enum Constant and Description

APPLICATION_MODAL

Defines a modal window that blocks events from being delivered to any other application window.

NONE

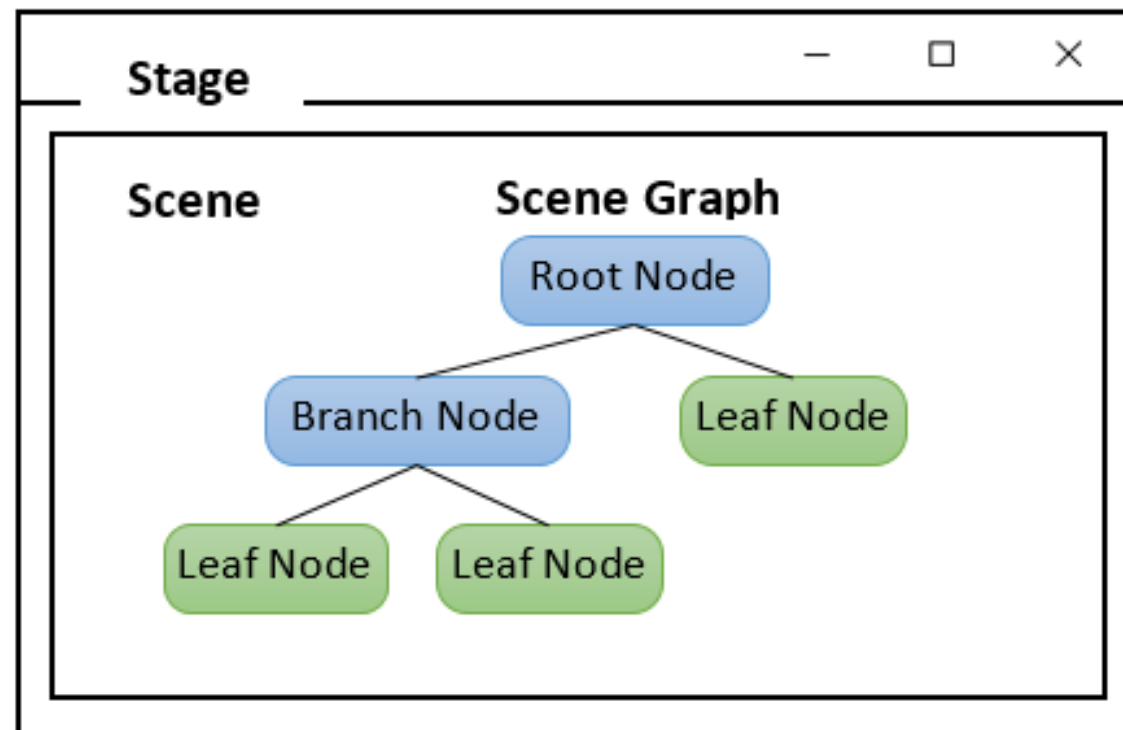
Defines a top-level window that is not modal and does not block any other window.

WINDOW_MODAL

Defines a modal window that block events from being delivered to its entire owner window hierarchy.

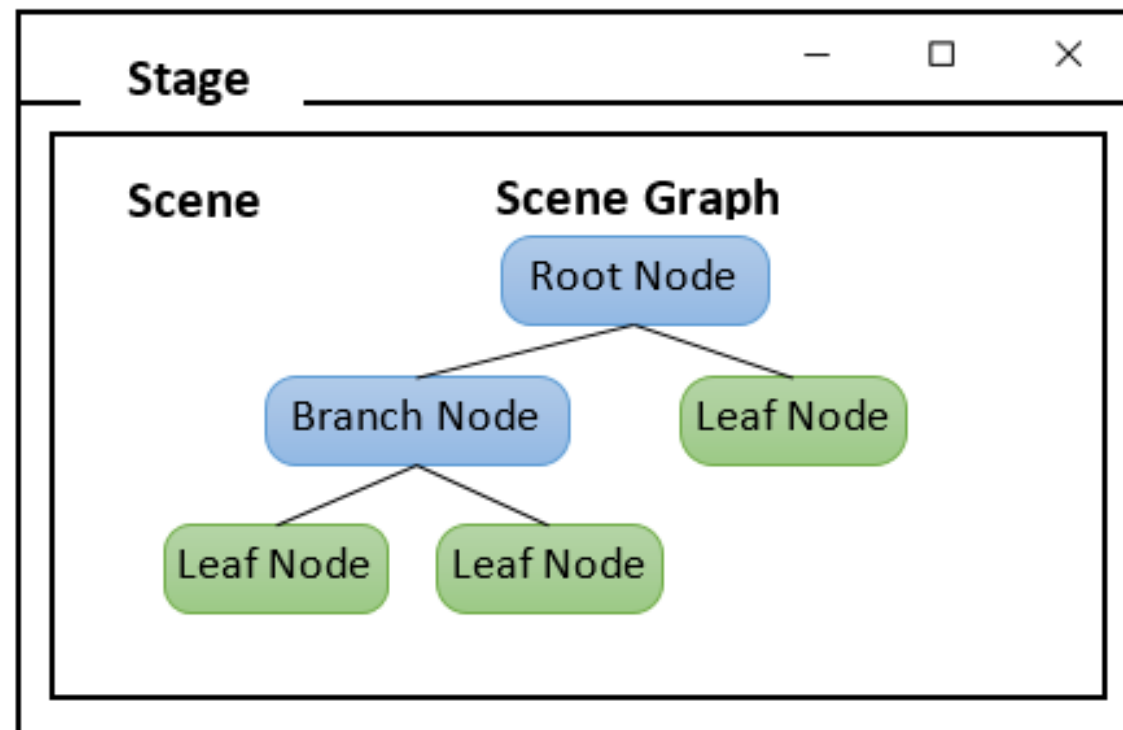
JavaFX Scene

- A JavaFX Scene contains all the visual JavaFX GUI components inside it
- A JavaFX Scene object is created by specifying a root GUI component (root node in the Scene Graph)
- A JavaFX Scene must be set on a JavaFX Stage to be visible
- A Scene can be attached to only a single Stage at a time, and Stage can also only display one Scene at a time.

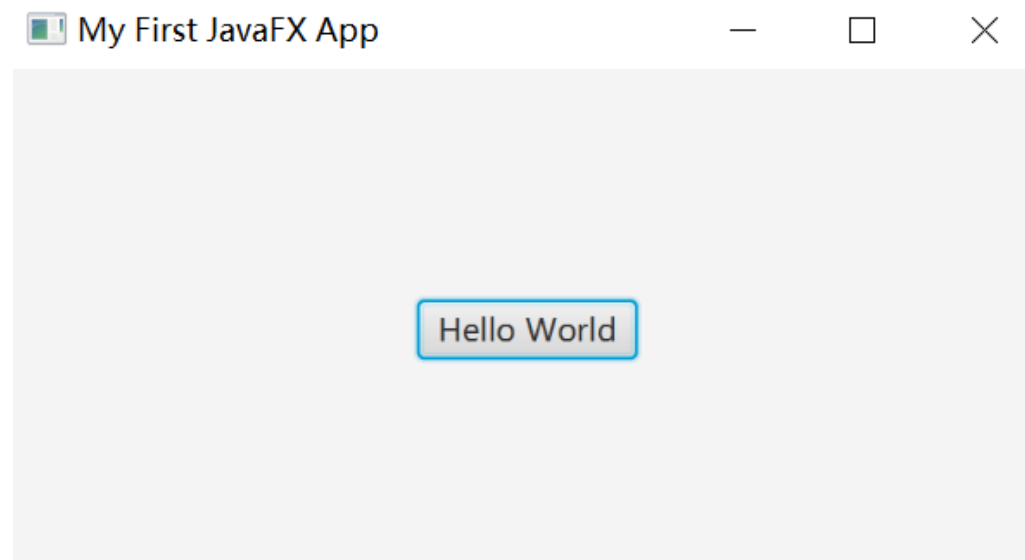
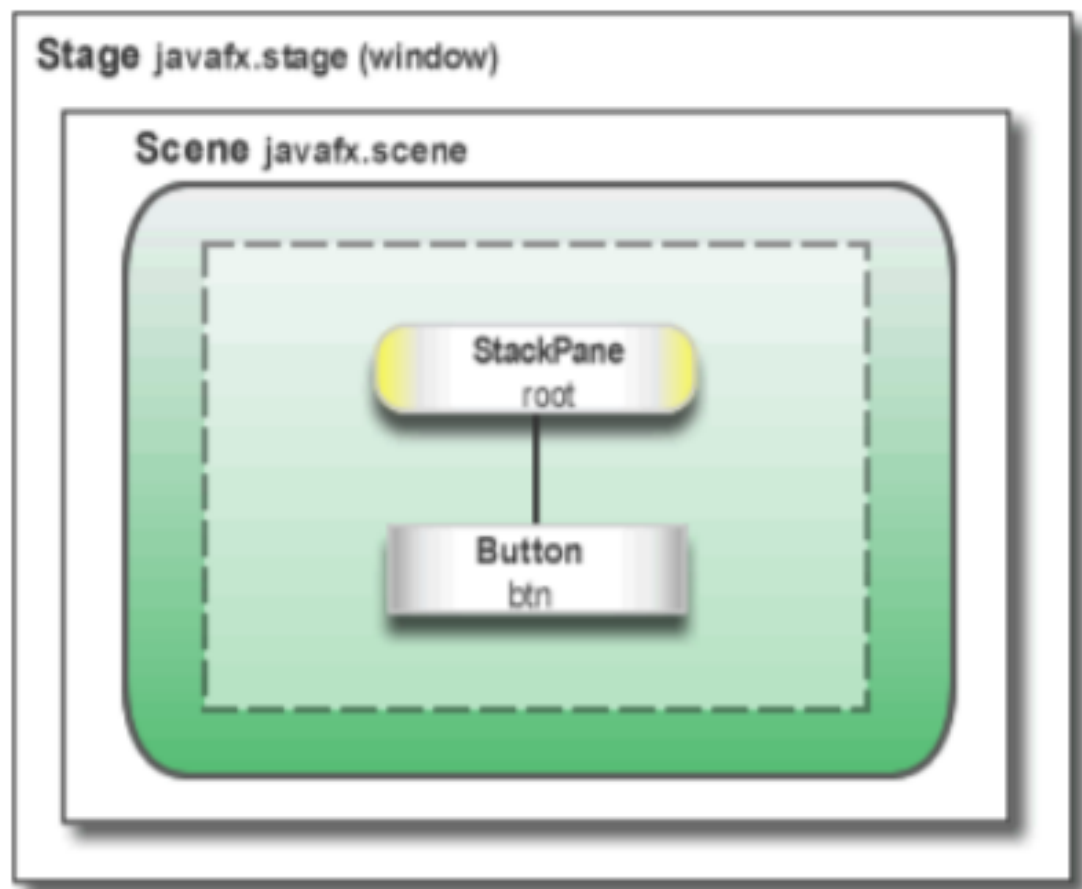


Scene Graph

- A tree data structure of **nodes**
- A node is a visual object of a JavaFX application
- Each node is classified as either a **branch node** (it can have children), or a **leaf node** (it cannot have children)
- A JavaFX application must specify the root node for the scene graph by setting the root property.



JavaFX Hello World



```

@Override
public void start(Stage primaryStage) throws Exception {
    primaryStage.setTitle("My First JavaFX App");

    StackPane root = new StackPane();

    Button btn = new Button();
    btn.setText("Hello World");
    btn.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) {
            System.out.println("Hello World!");
        }
    });

    root.getChildren().add(btn);

    Scene scene = new Scene(root, width: 400, height: 200);
    primaryStage.setScene(scene);

    primaryStage.show();
}

```

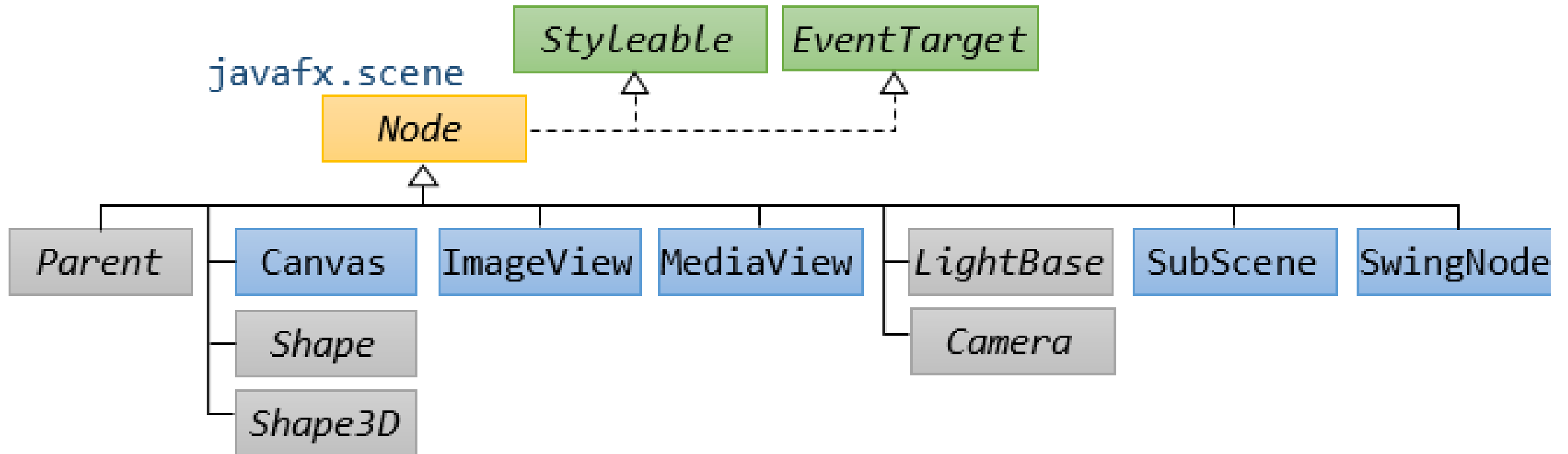
JavaFX Hello World

1. The root node is a StackPane object, a resizable layout node
2. The child node is a Button object, with an event handler for printing a message when pressed
3. Add button to the root node
4. Create a scene with the root
5. Set the scene for the stage and show

Node

A node is defined by an abstract class `javafx.scene.Node`, which is the superclass of all the UI elements

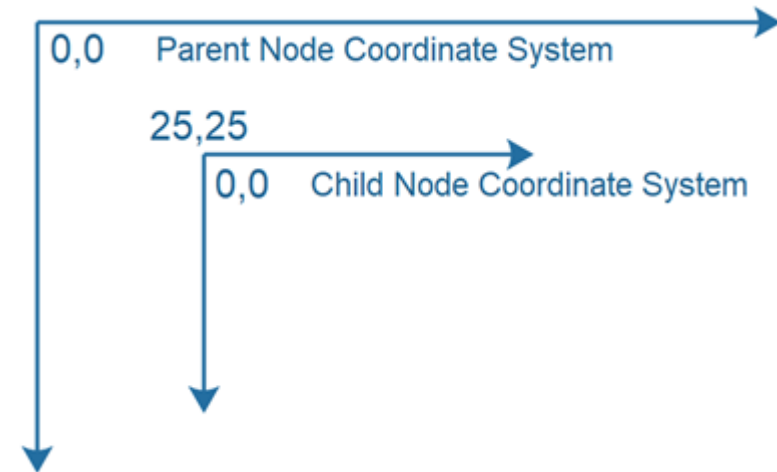
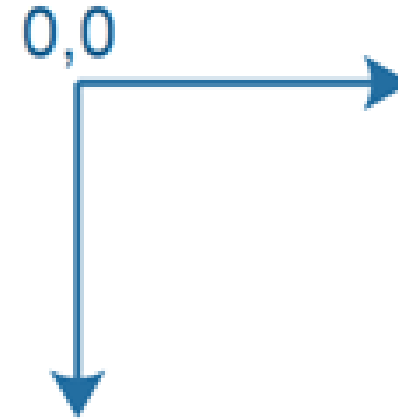
https://www3.ntu.edu.sg/home/ehchua/programming/java/Javafx1_intro.html



JavaFX Node Coordinate System (坐标系统)

- Each JavaFX Node has its own coordinate system.
- Difference from regular coordinate system: Y axis is reversed
- Use the coordinates to position child Node instances within the parent Node (see layoutX, layoutY)

<http://tutorials.jenkov.com/javafx/node.html>



JavaFX Node Property

(Writable) properties include X and Y position, width and height, text, children, event handlers, etc.

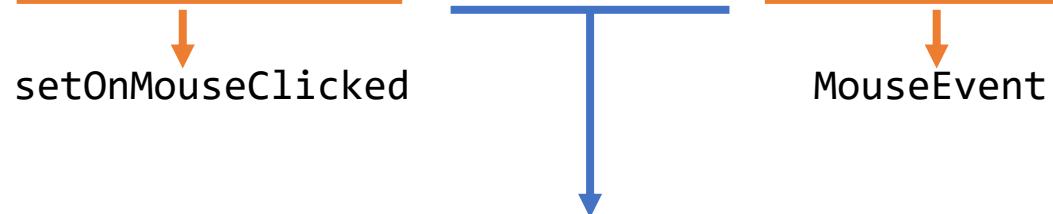
<code>ReadOnlyBooleanProperty</code>	<code>focused</code> Indicates whether this Node currently has focus.	<code>DoubleProperty</code>	<code>opacity</code> Specifies how opaque (that is, solid) the Node appears.
<code>BooleanProperty</code>	<code>focusTraversable</code> Specifies whether this Node should be a focusable node.	<code>ReadOnlyObjectProperty<Parent></code>	<code>parent</code> The parent of this Node.
<code>ReadOnlyBooleanProperty</code>	<code>hover</code> Whether or not this Node is being hovered.	<code>BooleanProperty</code>	<code>pickOnBounds</code> Defines how the picking computation is done for this node when it is hit.
<code>StringProperty</code>	<code>id</code> The id of this Node.	<code>ReadOnlyBooleanProperty</code>	<code>pressed</code> Whether or not the Node is pressed.
<code>ObjectProperty<InputMethodRequests></code>	<code>inputMethodRequests</code> Property holding InputMethodRequests.	<code>DoubleProperty</code>	<code>rotate</code> Defines the angle of rotation about the Node's center, measured in degrees.
<code>ReadOnlyObjectProperty<Bounds></code>	<code>layoutBounds</code> The rectangular bounds that should be used for layout.	<code>ObjectProperty<Point3D></code>	<code>rotationAxis</code> Defines the axis of rotation of this Node.
<code>DoubleProperty</code>	<code>layoutX</code> Defines the x coordinate of the translation.	<code>DoubleProperty</code>	<code>scaleX</code> Defines the factor by which coordinates are scaled about the center.
<code>DoubleProperty</code>	<code>layoutY</code> Defines the y coordinate of the translation.	<code>DoubleProperty</code>	<code>scaleY</code> Defines the factor by which coordinates are scaled about the center.
		<code>DoubleProperty</code>	<code>scaleZ</code> Defines the factor by which coordinates are scaled about the center.

JavaFX Node EventHandler Property

Node contains various Event Handler properties which can be set to user defined Event Handlers using the setter methods

Setter Naming Convention

setOnTargetType(EventHandler<TargetEvent> v)



onKeyPressed

Defines a function to be called

onKeyReleased

Defines a function to be called

onKeyTyped

Defines a function to be called

onMouseClicked

Defines a function to be called

onMouseDragEntered

Defines a function to be called

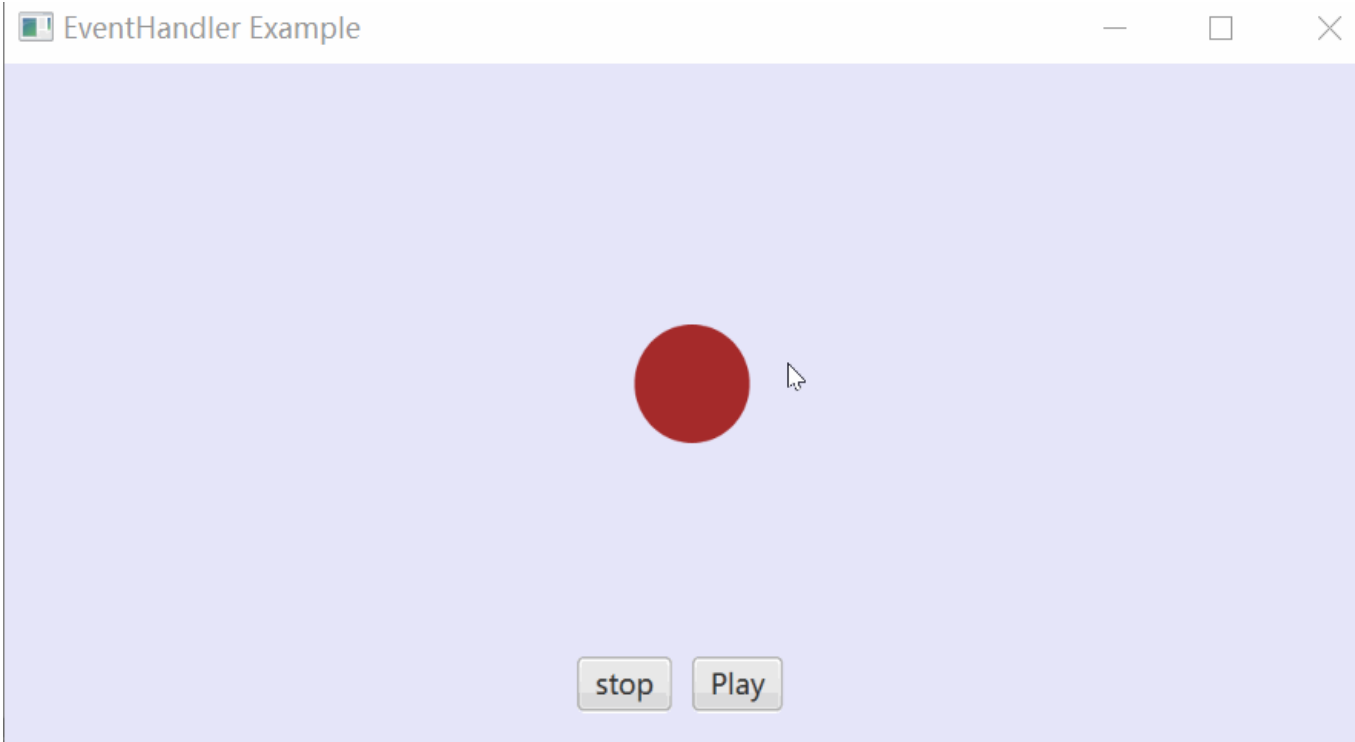
onMouseDragExited

Defines a function to be called

onMouseDragged

Defines a function to be called

How many events? What event handlers on which target?



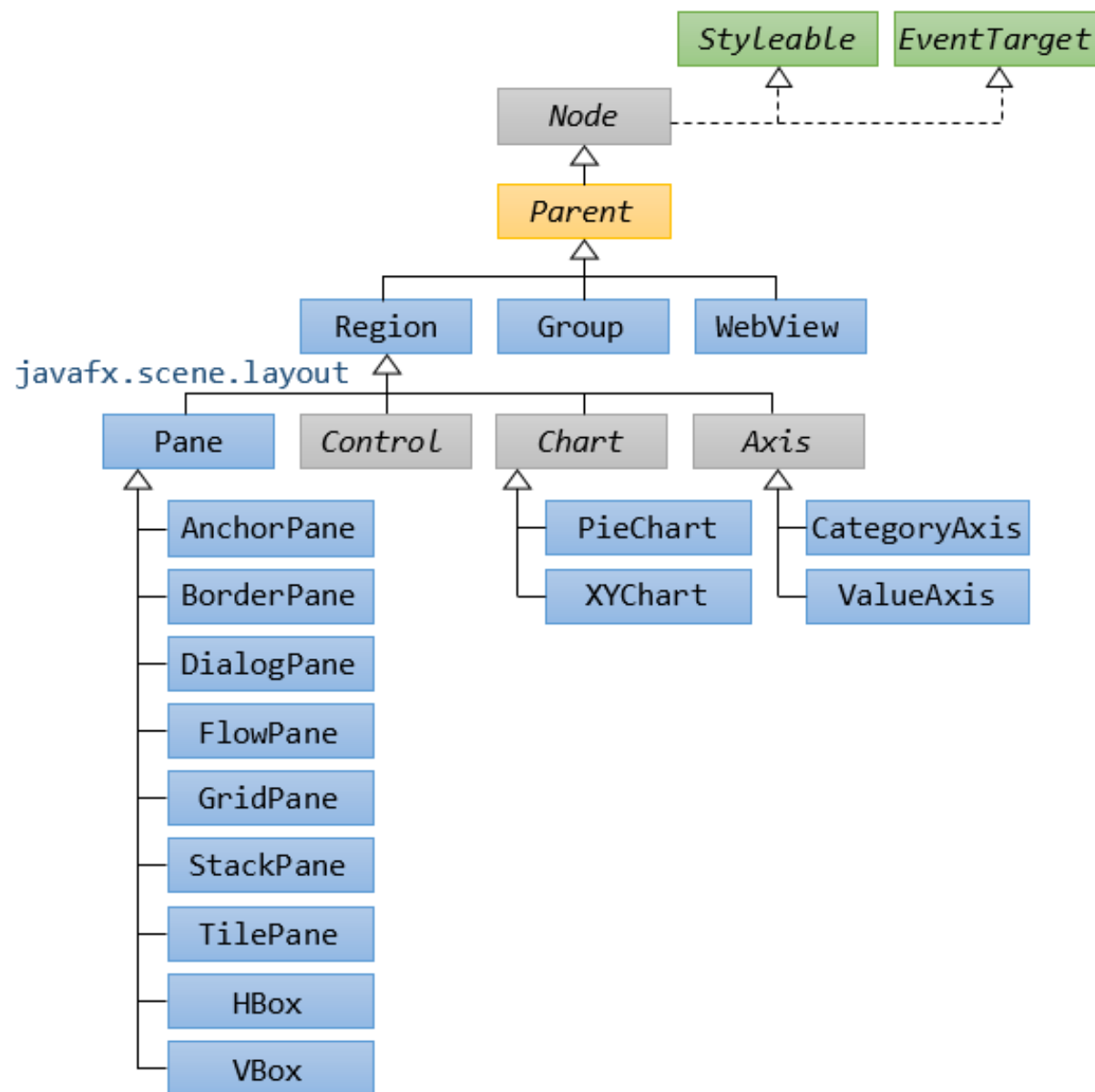
```
circle.setOnMouseClicked (new EventHandler<MouseEvent>() {  
    @Override  
    public void handle(javaafx.scene.input.MouseEvent e) {  
        circle.setFill(Color.DARKSLATEBLUE);  
    }  
});  
playButton.setOnMouseClicked((new EventHandler<MouseEvent>() {  
    public void handle(MouseEvent event) {  
        pathTransition.play();  
    }  
}));
```

```
stopButton.setOnMouseClicked((new EventHandler<MouseEvent>() {  
    public void handle(MouseEvent event) {  
        pathTransition.stop();  
    }  
}));
```

Full example code: https://www.tutorialspoint.com/javafx/javafx_event_handling.htm

Branch Node

- **Branch Node** (Parent): having child nodes. Defined by the abstract class `javafx.scene.Parent` with 3 concrete subclasses:
 - **Group**: Any transform (e.g. rotation), effect, or state applied to a Group will be applied to all children of that group.
 - **Region**: Region is the base class for all UI Controls and layout containers.
 - **WebView**: for HTML content.
- **Leaf Node**



JavaFX Layout

- Top-level container that organizes nodes in the scene graph
- `javafx.scene.layout` package provides various classes that represent the layouts
- `javafx.scene.layout.Pane` class is the parent class for all these built-in layout classes

```
Pane canvas = new Pane();
canvas.setStyle("-fx-background-color: black;");
canvas.setPrefSize(200,200);
Circle circle = new Circle(50,Color.BLUE);
circle.relocate(20, 20);
Rectangle rectangle = new Rectangle(100,100,Color.RED);
rectangle.relocate(70,70);
canvas.getChildren().addAll(circle,rectangle);
```

Pane (JavaFX 8) - Oracle

<https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/Pane.html>

Pane resizes each managed child regardless of the child's visible property value; unmanaged children are ignored for all layout calculations. Resizable Range A pane's parent will resize the...

GridPane

`javafx.geometry.Insets` Margin space around the outside of the child. By ...

BorderPane

A border pane's unbounded maximum width and height are an indication to the parent ...

StackPane

`javafx.scene.layout.Pane;`
`javafx.scene.layout.StackPane;` All ...

TilePane

TilePane (JavaFX 8) - Oracle. children - The initial set of children for this pane. Since: ...

FlowPane

FlowPane (JavaFX 8) - Oracle. children - The initial set of children for this pane. Since: ...

HBox

HBox (JavaFX 8) - Oracle. children - The initial set of children for this pane. Since: ...

VBox

VBox (JavaFX 8) - Oracle. children - The initial set of children for this pane. Since: ...

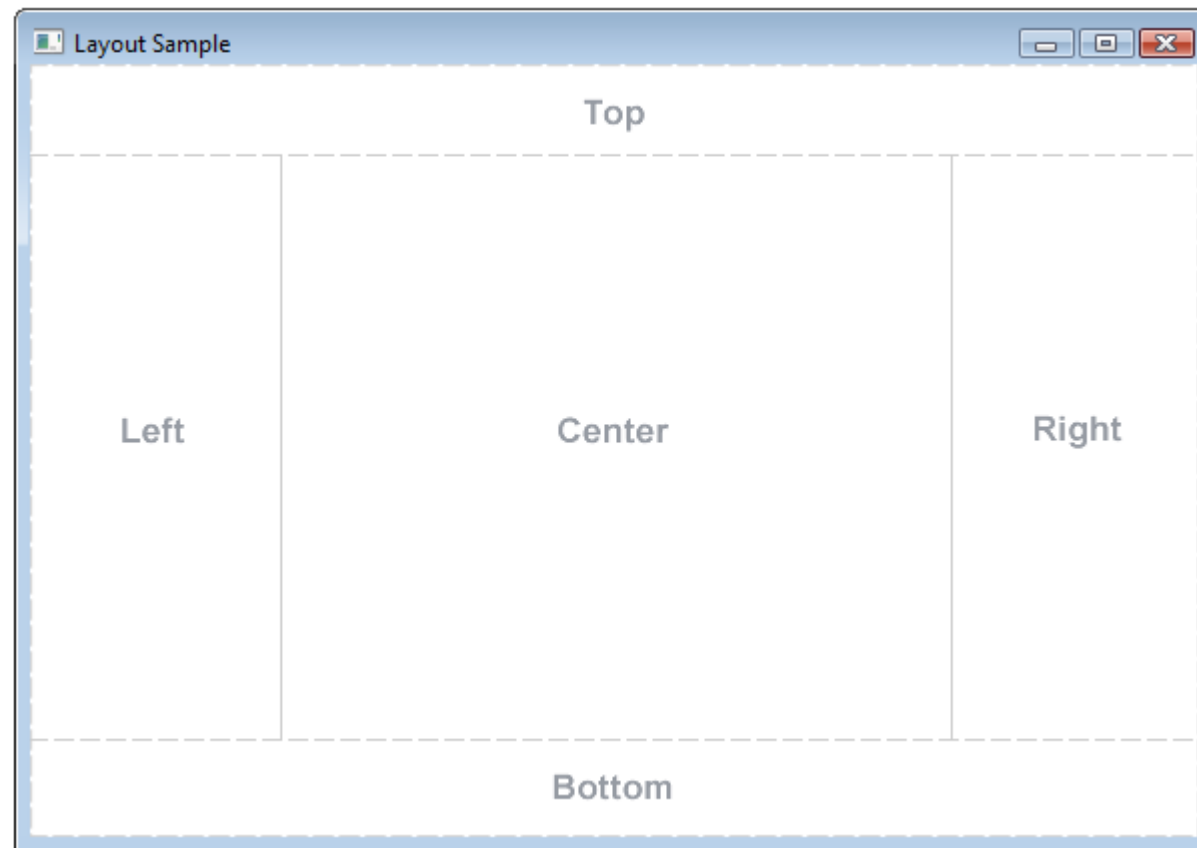
BorderPane

The BorderPane layout pane provides five regions in which to place nodes: top, bottom, left, right, and center.

For more details:

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

Figure 1-1 Sample Border Pane



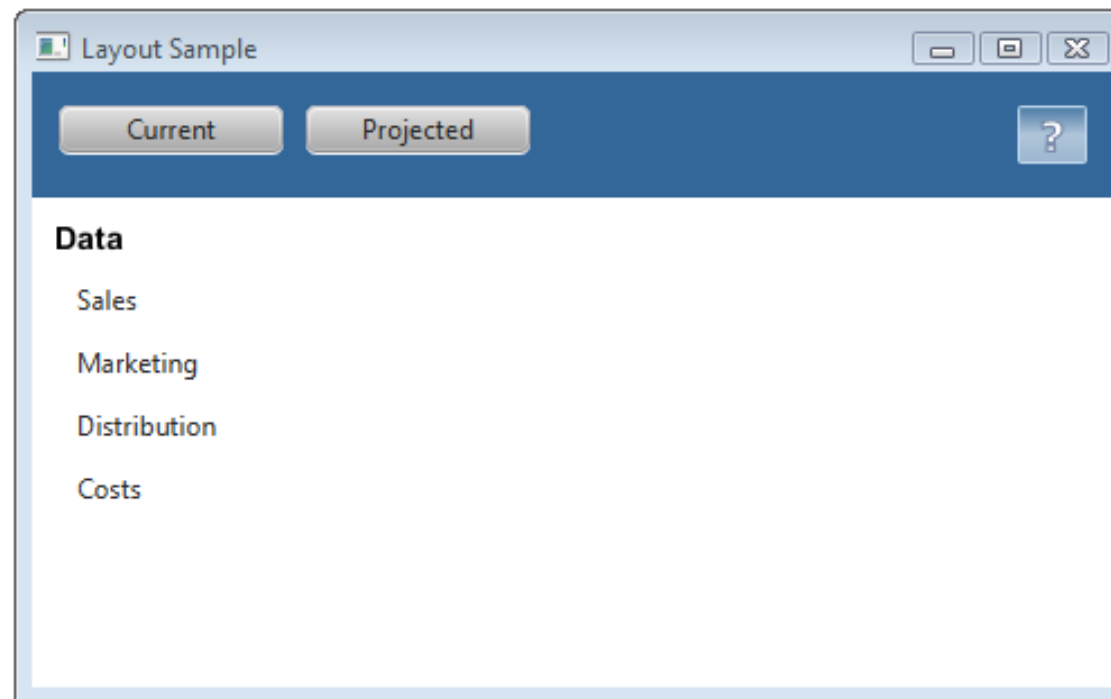
HBox & VBox Pane

- The HBox layout pane provides an easy way for arranging a series of nodes in a single row
- The VBox layout pane provides an easy way for arranging a series of nodes in a single column

For more details:

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

Figure 1-5 VBox Pane in a Border Pane

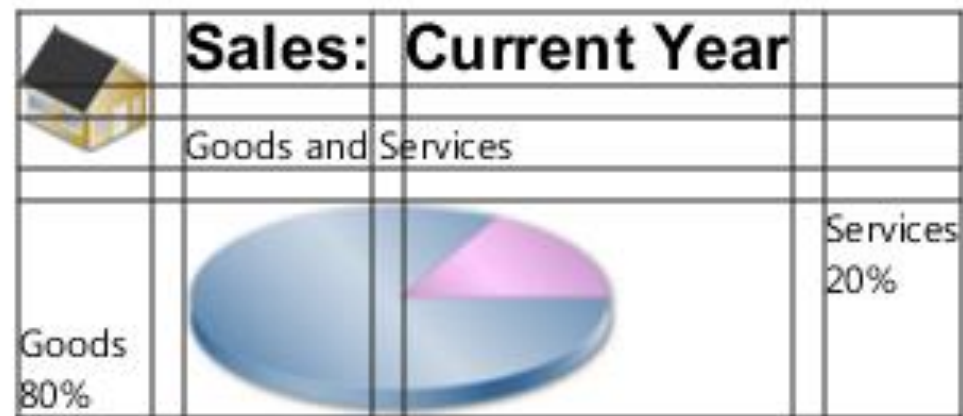


GridPane

The GridPane layout pane enables you to create a flexible grid of rows and columns in which to lay out nodes.

For more details:
https://docs.oracle.com/javafx/2/layout/builtin_layouts.htm

Figure 1-8 Sample Grid Pane



Combine Panes

Different Panes can be combined to make beautiful layout

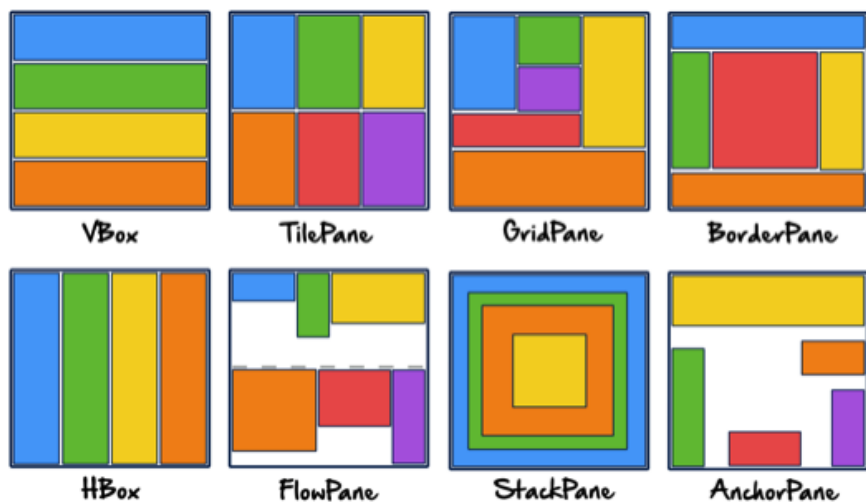
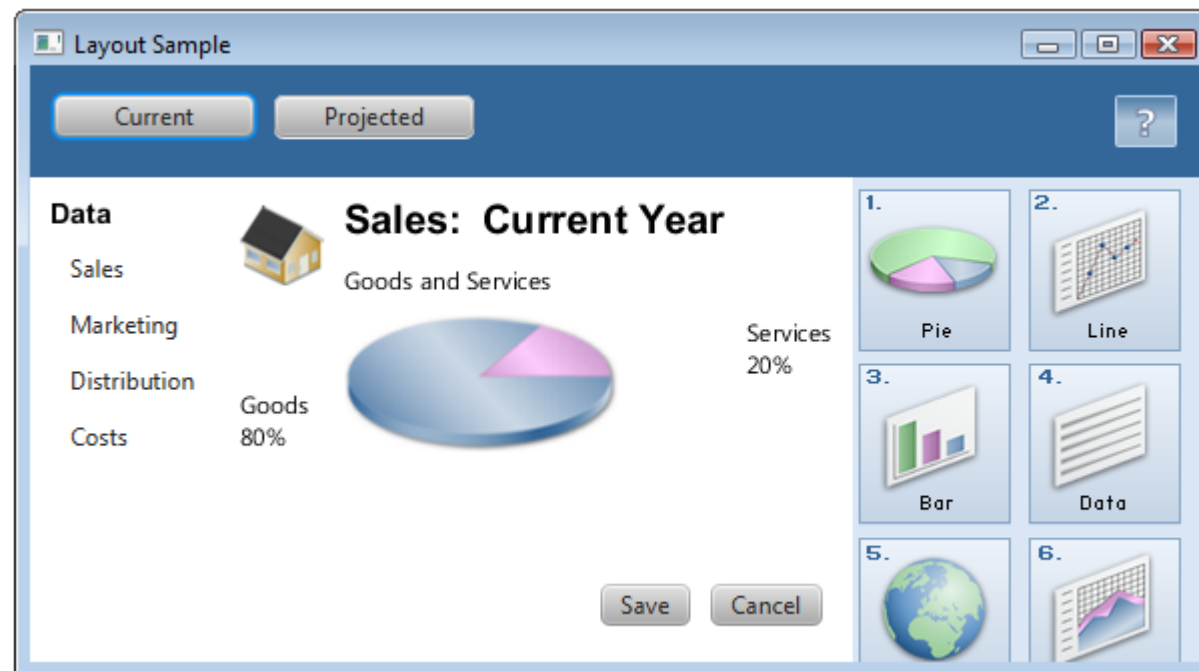
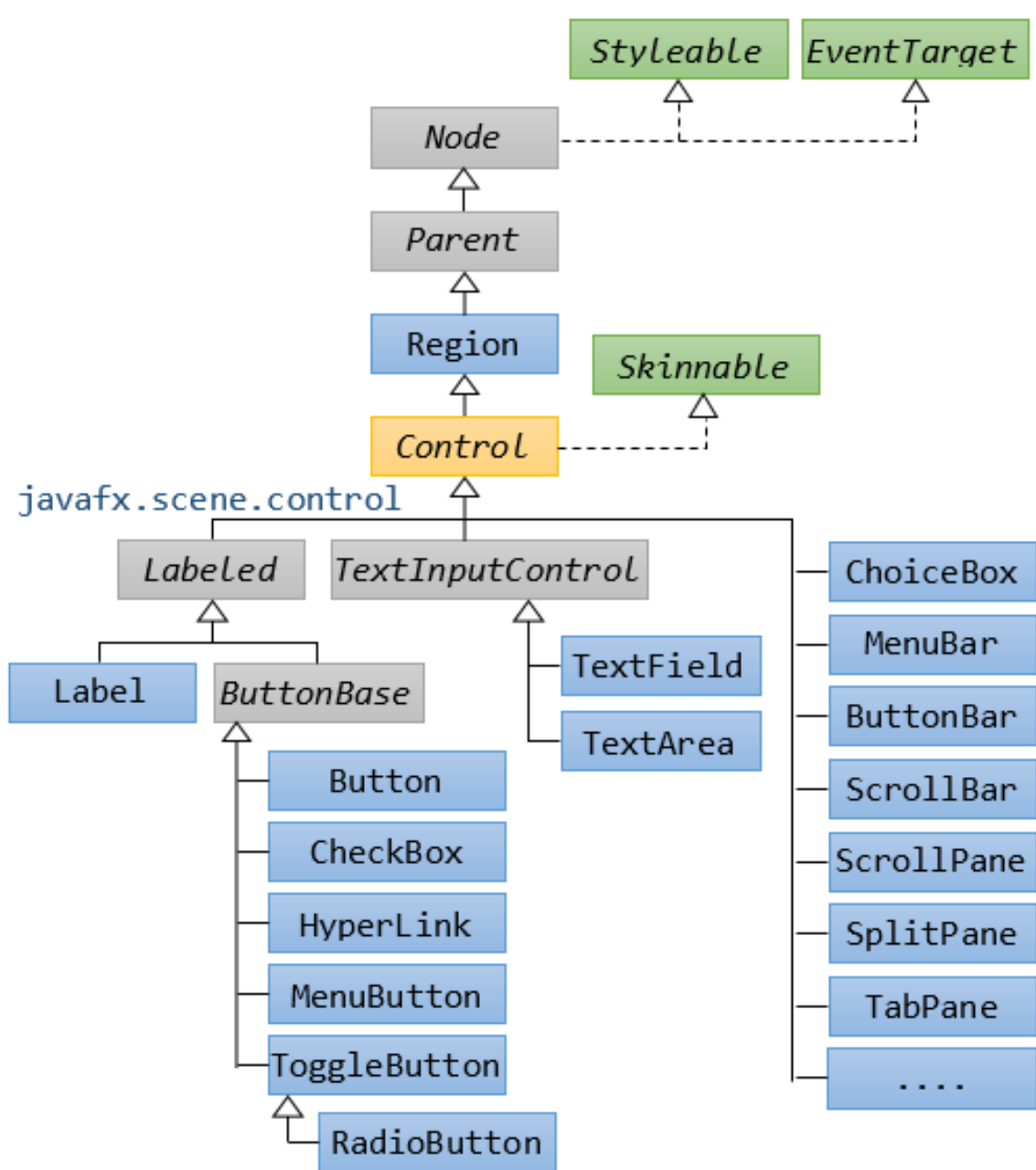


Image source: <https://dzone.com/refcardz/javafx-8-1>



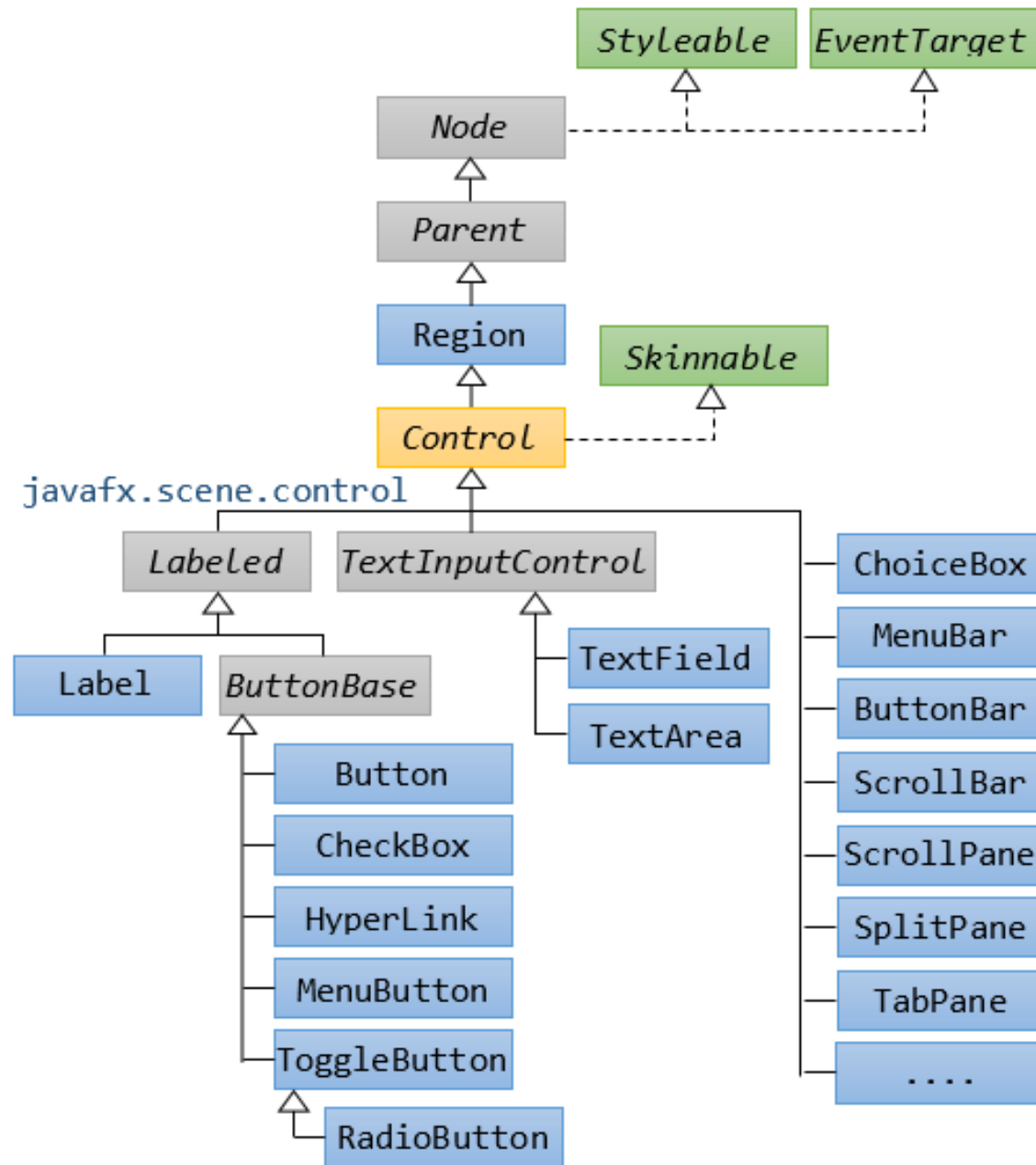
For more details:
https://docs.oracle.com/javafx/2/layout/builtin_layouts.htm

UI Controls

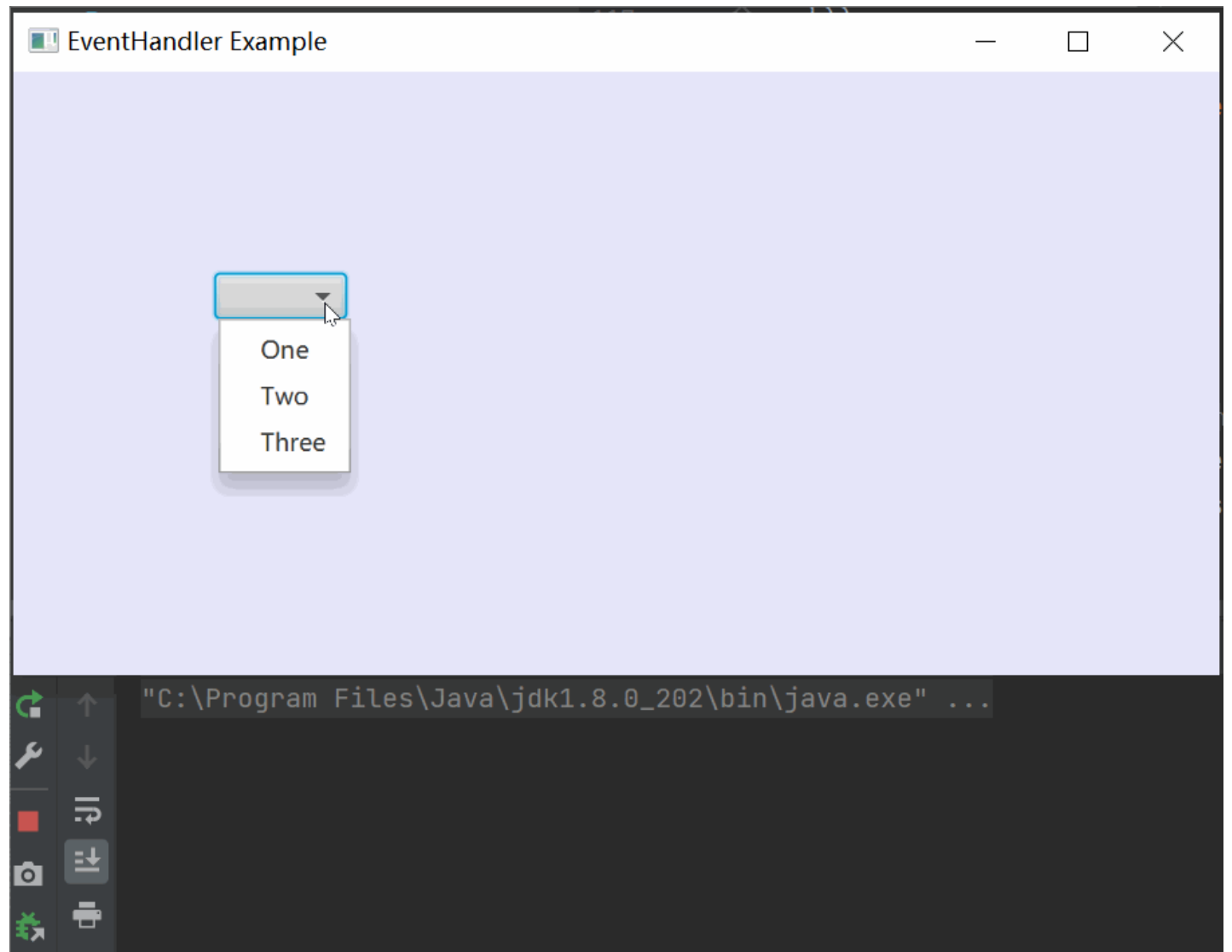


UI Controls

The **Skinnable** interface allows developers to separate the visual representation (skin) of a control from its behavior and logic.



Example: ChoiceBox



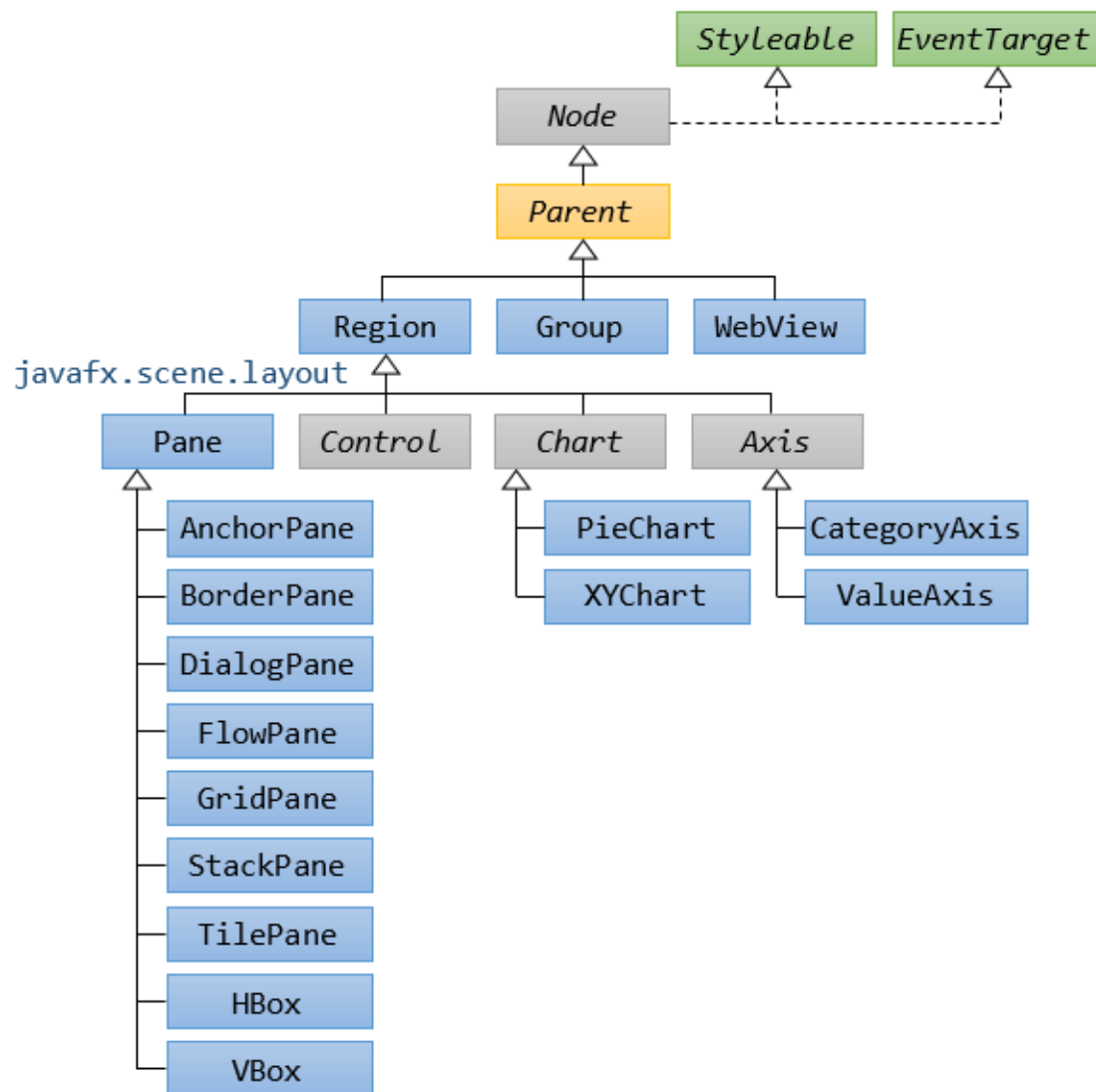
Example: ChoiceBox

Add a ChangeListener which will be notified whenever the value of the choicebox changes.

```
ChoiceBox<String> box = new ChoiceBox<String>();
box.setLayoutX(100);
box.setLayoutY(100);
box.getItems().add("One");
box.getItems().add("Two");
box.getItems().add("Three");

box.getSelectionModel().addChangeListener(
    new ReadOnlyObjectProperty<String>() {
        @Override public void addListener(
            ObservableValue<? extends String> observable,
            String oldValue, String newValue) {
            System.out.println(oldValue + "->" + newValue);
        }
    }
);
```

ChangeListener is functional interface, you can use lambda here



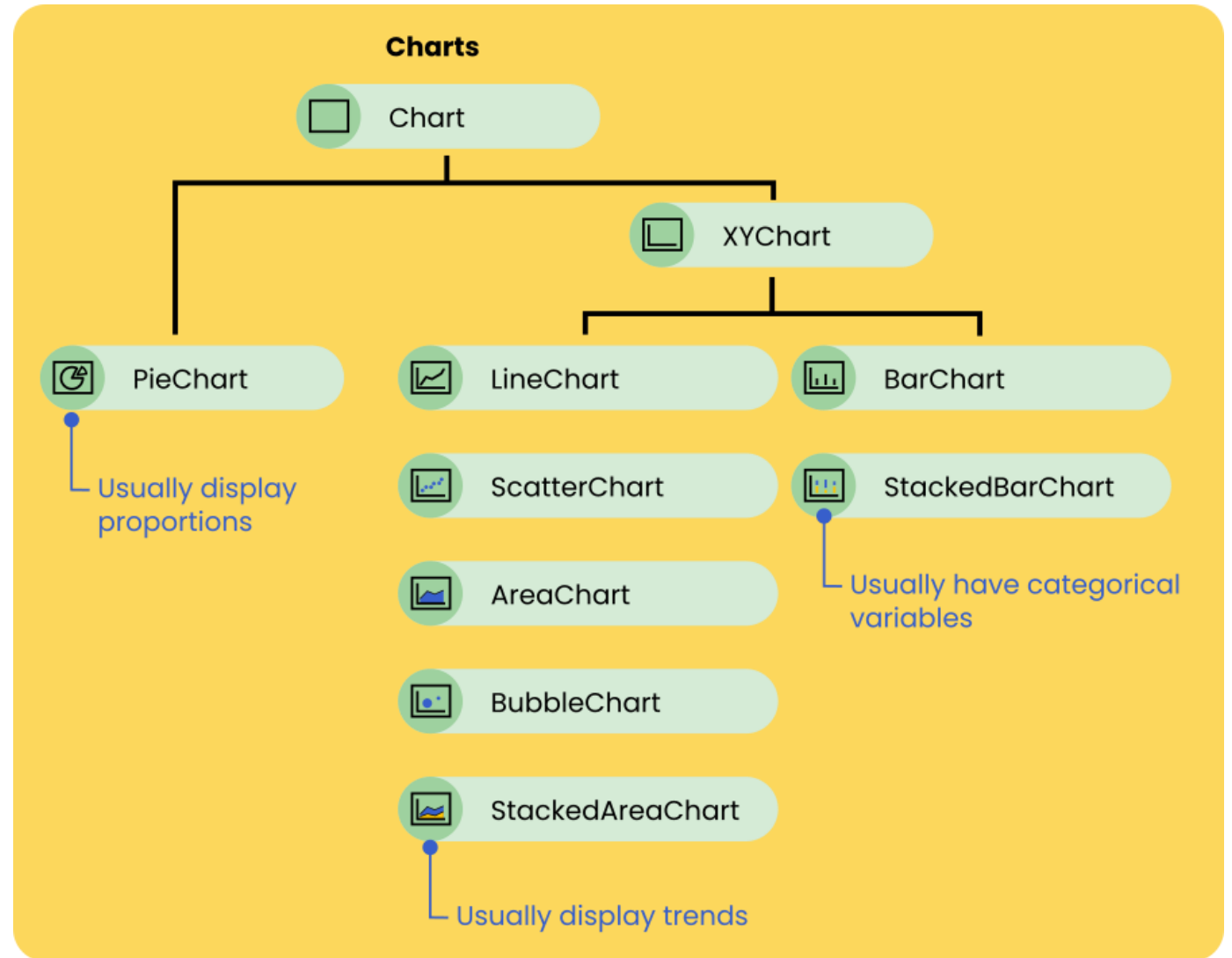
JavaFX Charts

- Chart: a graphical representation of data in the form of symbols
- JavaFX Chart (`javafx.scene.chart.Chart`) is the base class for all charts. It has 3 parts:
 - Title
 - Legend (图例)
 - `chartContent`

Types of Charts

JavaFX provides 8 default charts to display data, which fall in two types (PieChart & XYChart)

<https://edencoding.com/javafx-charts/>



PieChart (饼图)

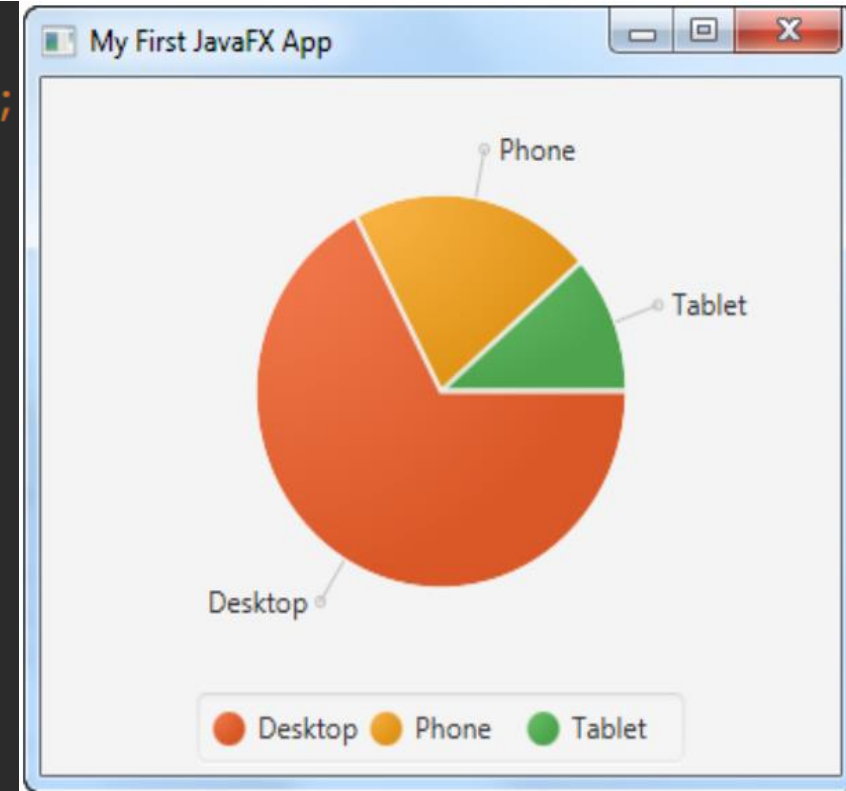
Works the best to find out the composition of something

PieChart.Data

PieChart Data Item, represents one slice in the PieChart

```
PieChart pieChart = new PieChart();
PieChart.Data slice1 = new PieChart.Data(name: "Desktop", value: 213);
PieChart.Data slice2 = new PieChart.Data(name: "Phone", value: 67);
PieChart.Data slice3 = new PieChart.Data(name: "Tablet", value: 36);
pieChart.getData().add(slice1);
pieChart.getData().add(slice2);
pieChart.getData().add(slice3);

VBox vbox = new VBox(pieChart);
Scene scene = new Scene(vbox, width: 400, height: 200);
primaryStage.setScene(scene);
primaryStage.show();
```



LineChart (折线图)

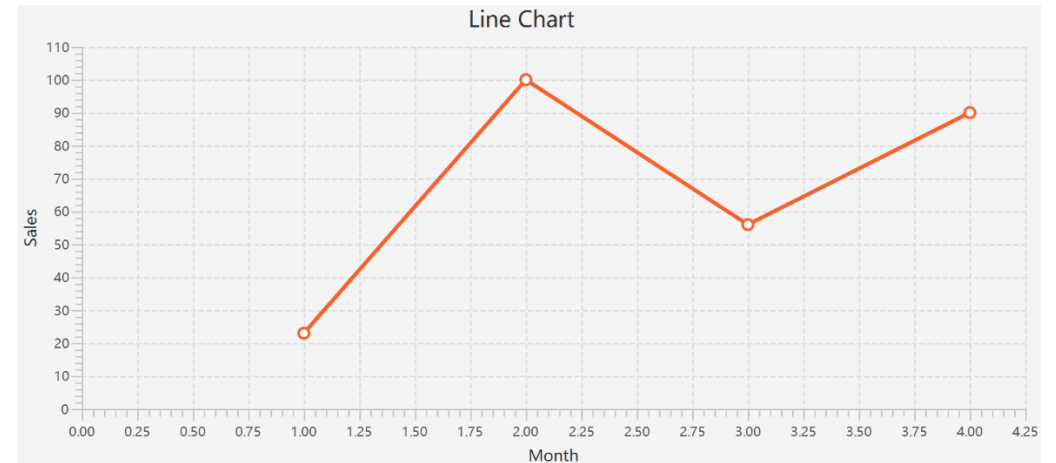
most often used to visualize data that changes over time

```
NumberAxis xAxis = new NumberAxis();
NumberAxis yAxis = new NumberAxis();
LineChart<Number, Number> lineChart = new LineChart<>(xAxis, yAxis);

lineChart.setTitle("Line Chart");
xAxis.setLabel("Month");
yAxis.setLabel("Sales");

XYChart.Series<Number, Number> series = new XYChart.Series<>();
series.getData().add(new XYChart.Data<>( xValue: 1, yValue: 23));
series.getData().add(new XYChart.Data<>( xValue: 2, yValue: 100));
series.getData().add(new XYChart.Data<>( xValue: 3, yValue: 56));
series.getData().add(new XYChart.Data<>( xValue: 4, yValue: 90));

Scene scene = new Scene(lineChart, width: 800, height: 400);
lineChart.getData().add(series);
stage.setScene(scene);
stage.show();
```

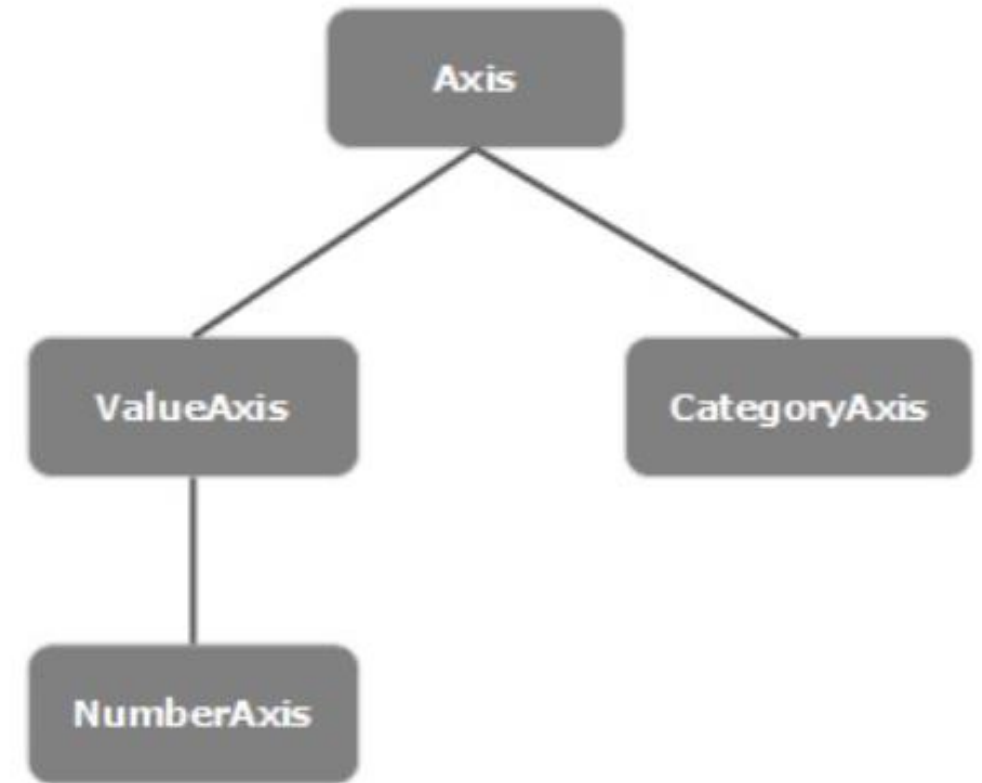


Nested Classes

Modifier and Type	Class and Description
static class	XYChart.Data<X, Y> A single data item with data for 2 axis charts
static class	XYChart.Series<X, Y> A named series of data items

Axis

- An abstract class representing X or Y axis
- NumberAxis
 - Quantity, Age, Population, etc.
- CategoryAxis
 - Countries, Weekdays, Colors, etc.



https://www.tutorialspoint.com/javafx/javafx_charts.htm

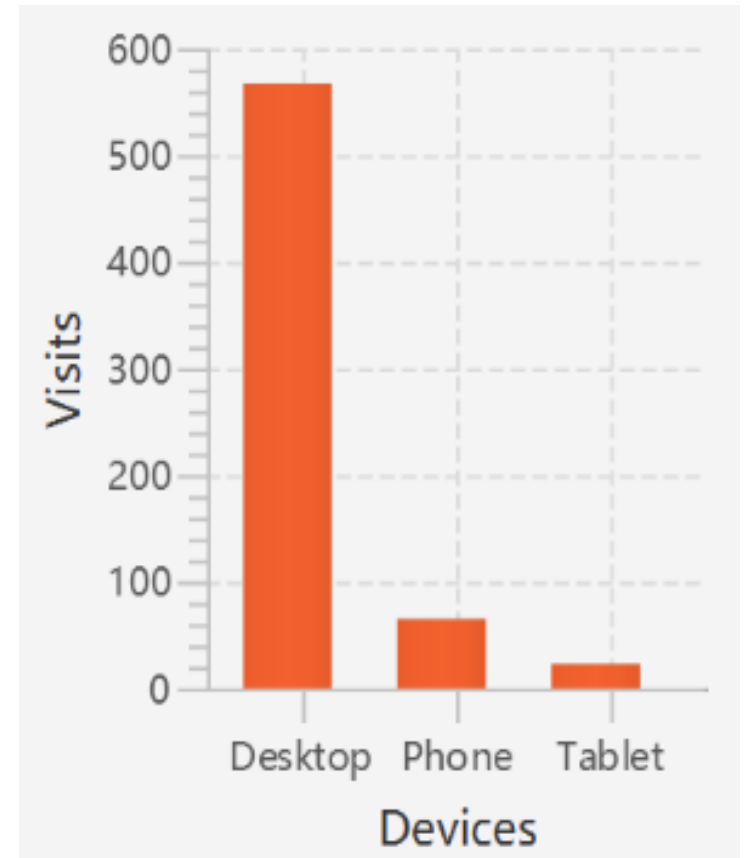
Using CategoryAxis

```
CategoryAxis xAxis = new CategoryAxis();
NumberAxis yAxis = new NumberAxis();
xAxis.setLabel("Devices");
yAxis.setLabel("Visits");

BarChart<String, Number> barChart = new BarChart<>(xAxis, yAxis);

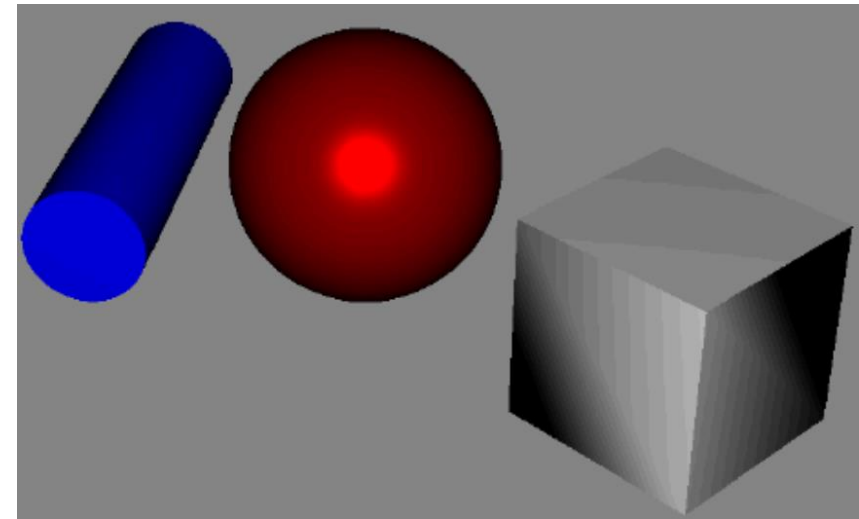
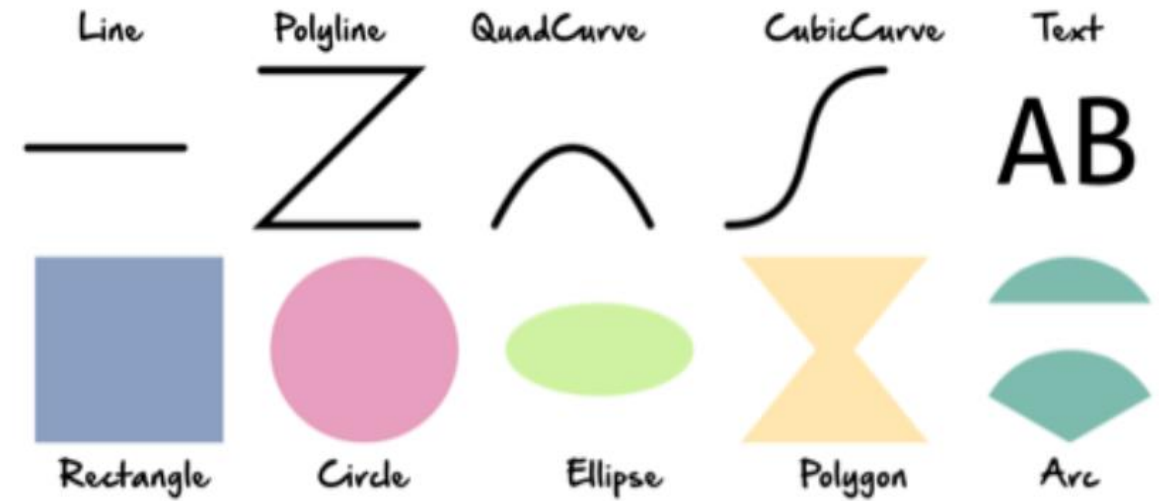
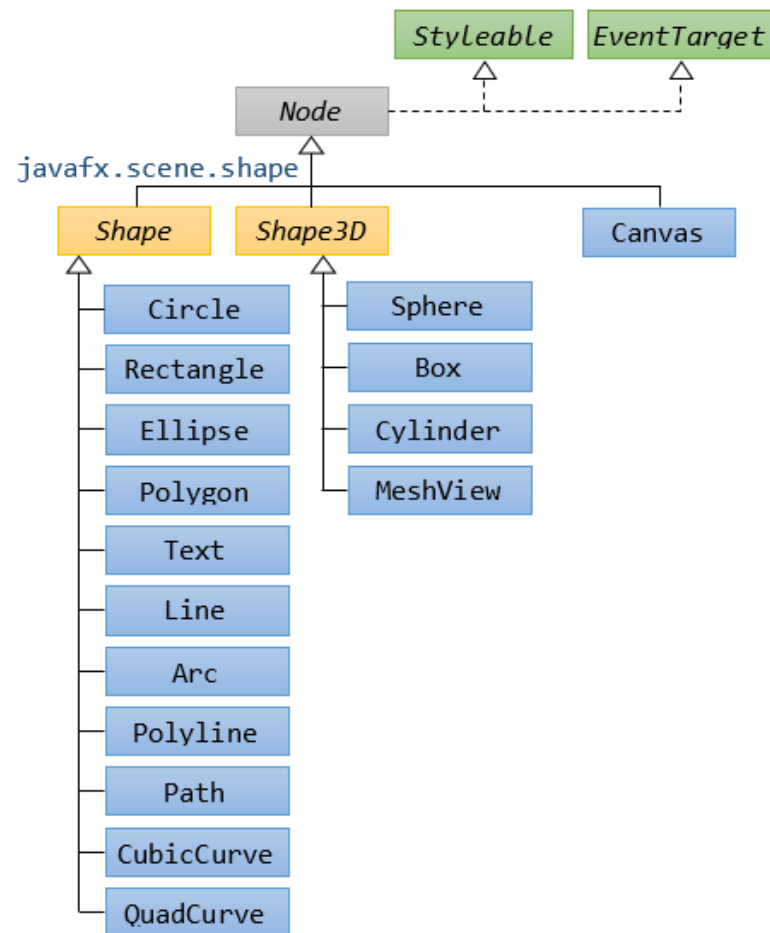
XYChart.Series<String, Number> data = new XYChart.Series<>();
data.getData().add(new XYChart.Data<>(xValue: "Desktop", yValue: 567));
data.getData().add(new XYChart.Data<>(xValue: "Phone", yValue: 65));
data.getData().add(new XYChart.Data<>(xValue: "Tablet", yValue: 23));

barChart.getData().add(data);
```



Full example: <http://tutorials.jenkov.com/javafx/barchart.html>

JavaFX Shape



Shape Properties

- Fill
- Stroke/Outline
- Decoration styles



Image source: <https://dzone.com/refcardz/javafx-8-1>



CENTERED



OUTSIDE



INSIDE



Color



LinearGradient



RadialGradient



ImagePattern

Shape Operations

We could use operations including intersect, union, and subtract to create new shapes

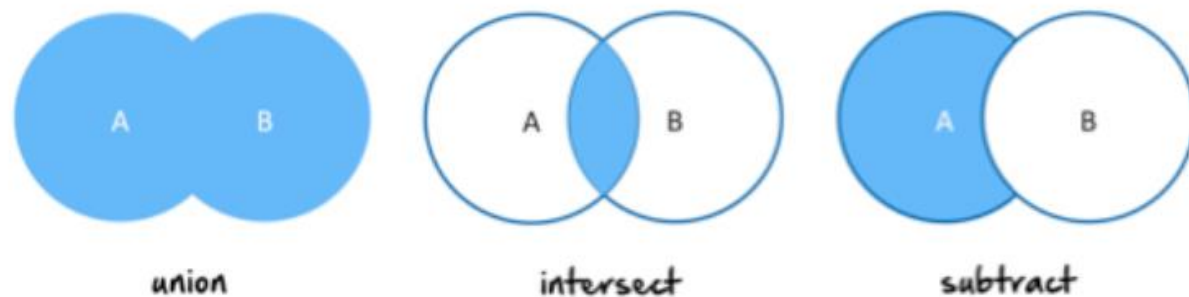


Image source: <https://dzone.com/refcardz/javafx-8-1>

JavaFX Shape

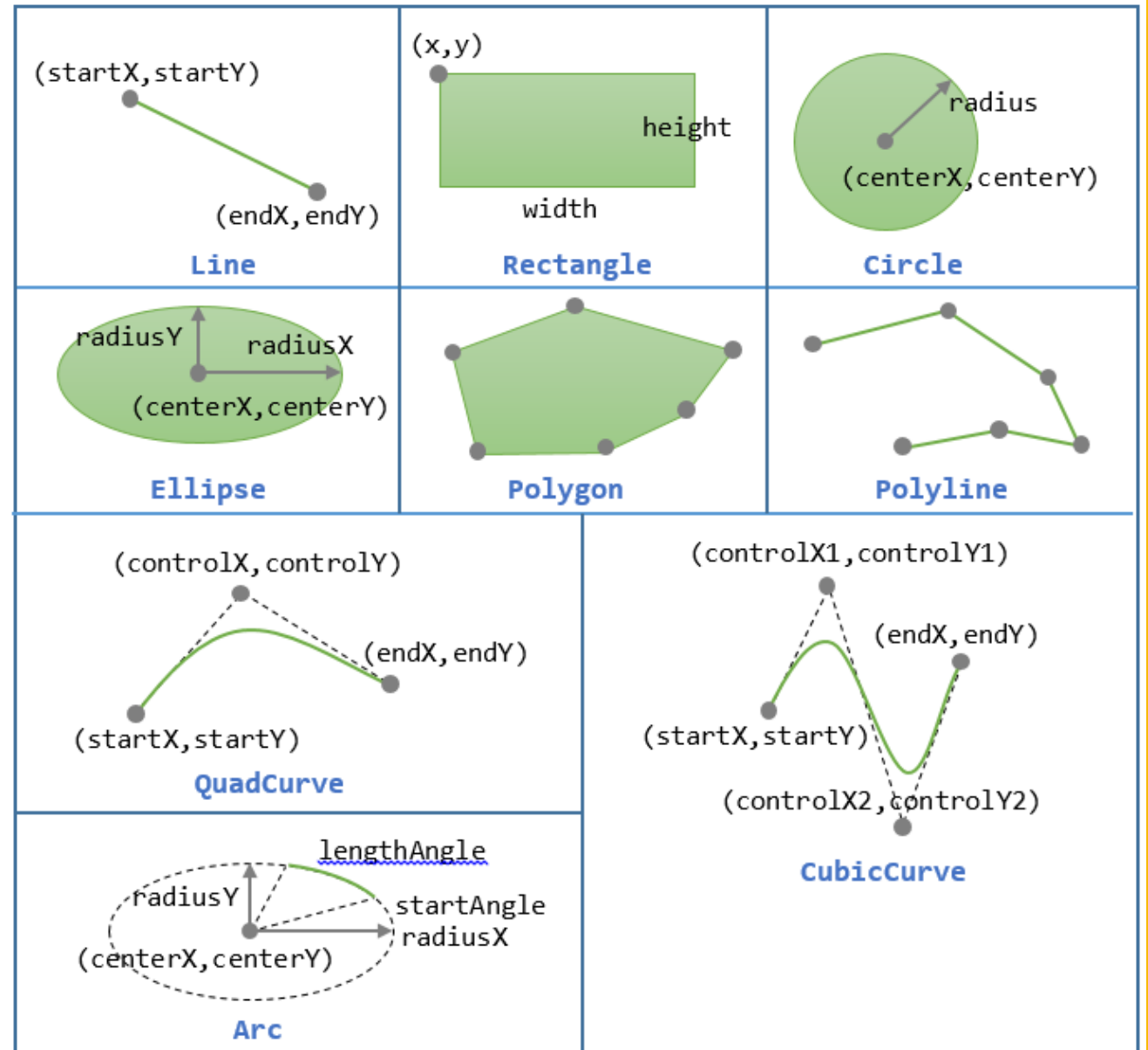
```
Circle circle = new Circle();

//Setting the position of the circle
circle.setCenterX(300.0f);
circle.setCenterY(135.0f);

//Setting the radius of the circle
circle.setRadius(25.0f);

//Setting the color of the circle
circle.setFill(Color.BROWN);

//Setting the stroke width of the circle
circle.setStrokeWidth(20);
```





Lecture 9

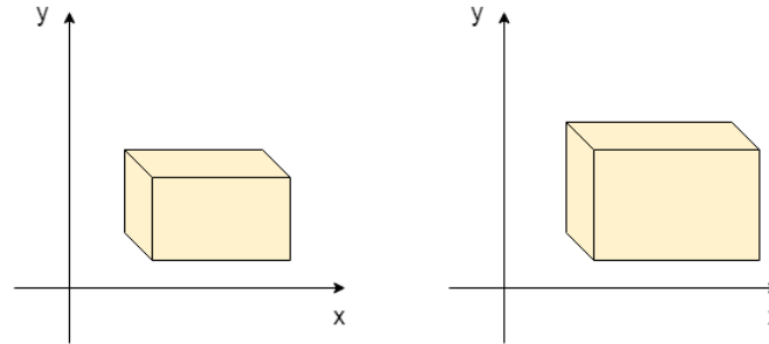
- JavaFX
 - Overview
 - Hello World
 - Design & Concepts
 - Layouts, Shapes, UI controls
 - Charts and Axis
 - Transformation, Animation, Effects
 - FXML
 - Multithreading in JavaFX

JavaFX Transformation

A transformation changes the place of a graphical object in a coordinate system according to certain parameters.

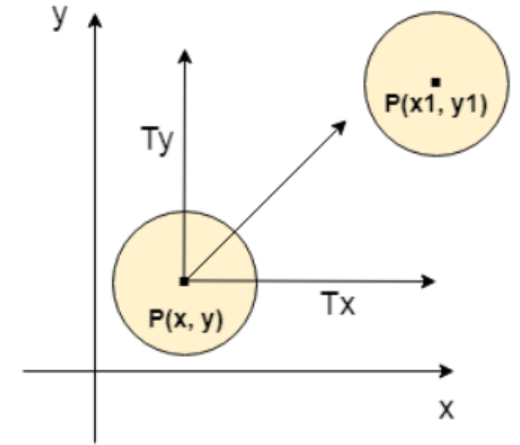
Source: <https://www.javatpoint.com/javafx-transformation>

TAO Yida@SUSTECH



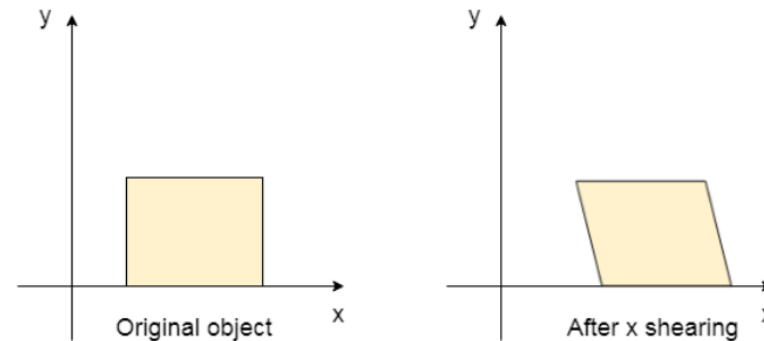
Scaling

Change the size of an object



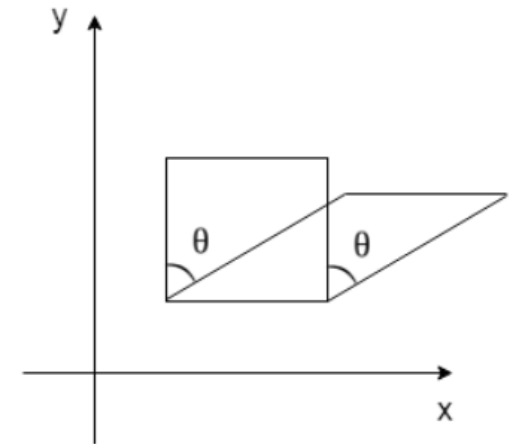
Translation

Change in the position of an object



Shearing

Change the slope of an object w.r.t. any axis

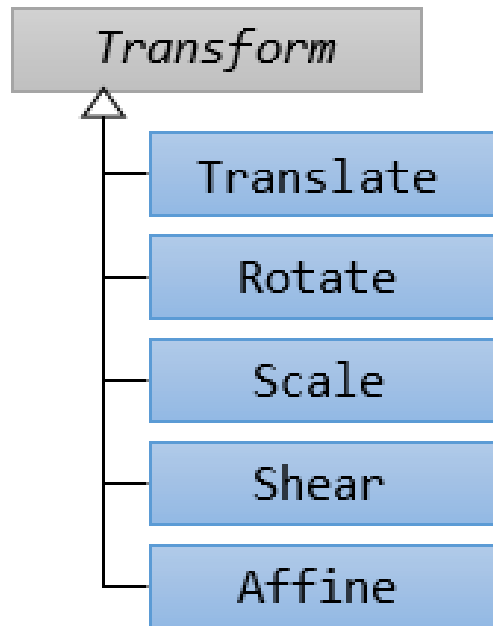


Rotation

Rotate an object by a certain angle θ

JavaFX Transformation

`javafx.scene.transform`

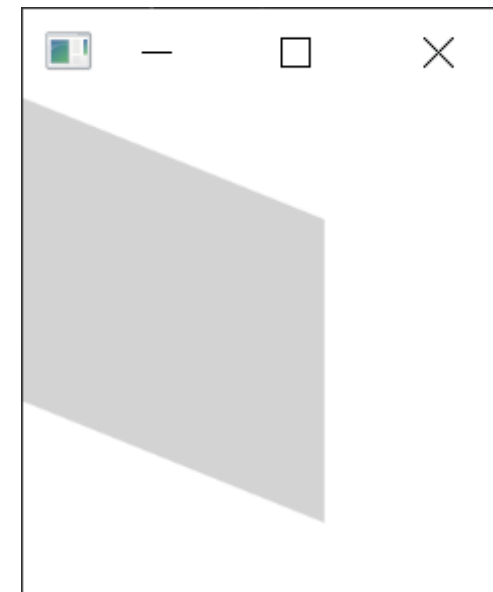
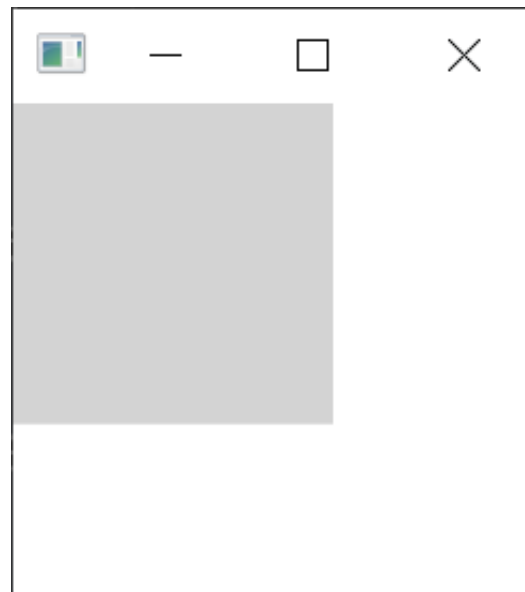


- All transformations are represented by various classes in package `javafx.scene.transform`
- `Transform` is the base class for different transformations

```
Rectangle rect = new Rectangle(50,50, Color.RED);  
rect.getTransforms().add(new Rotate(45,0,0)); //rotate by 45 degrees
```

Example

```
Group rectangleGroup = new Group();  
Rectangle rect = new Rectangle();  
  
Shear sh = new Shear();  
sh.setY(0.4);  
  
rect.getTransforms().add(sh);  
rectangleGroup.getChildren().add(rect);
```



JavaFX Effects

✿ SepiaTone



✿ Glow



✿ ColorAdjust



✿ Reflection

Reflections on JavaFX...
K6116C10U2 0U 19A91-Y"

not shown:

✿ ColorInput

✿ Shadow

✿ ImageInput

✿ Bloom

Bloom!

✿ Lighting

JavaFX!

✿ DropShadow

JavaFX drop shadow...
●

✿ InnerShadow

InnerShadow

✿ GaussianBlur

Blurry Text!

✿ BoxBlur

Blurry Text!

✿ MotionBlur

Motion

✿ javafx.scene.effect

✿ DisplacementMap

Wavy Text

✿ PerspectiveTransform

Perspective

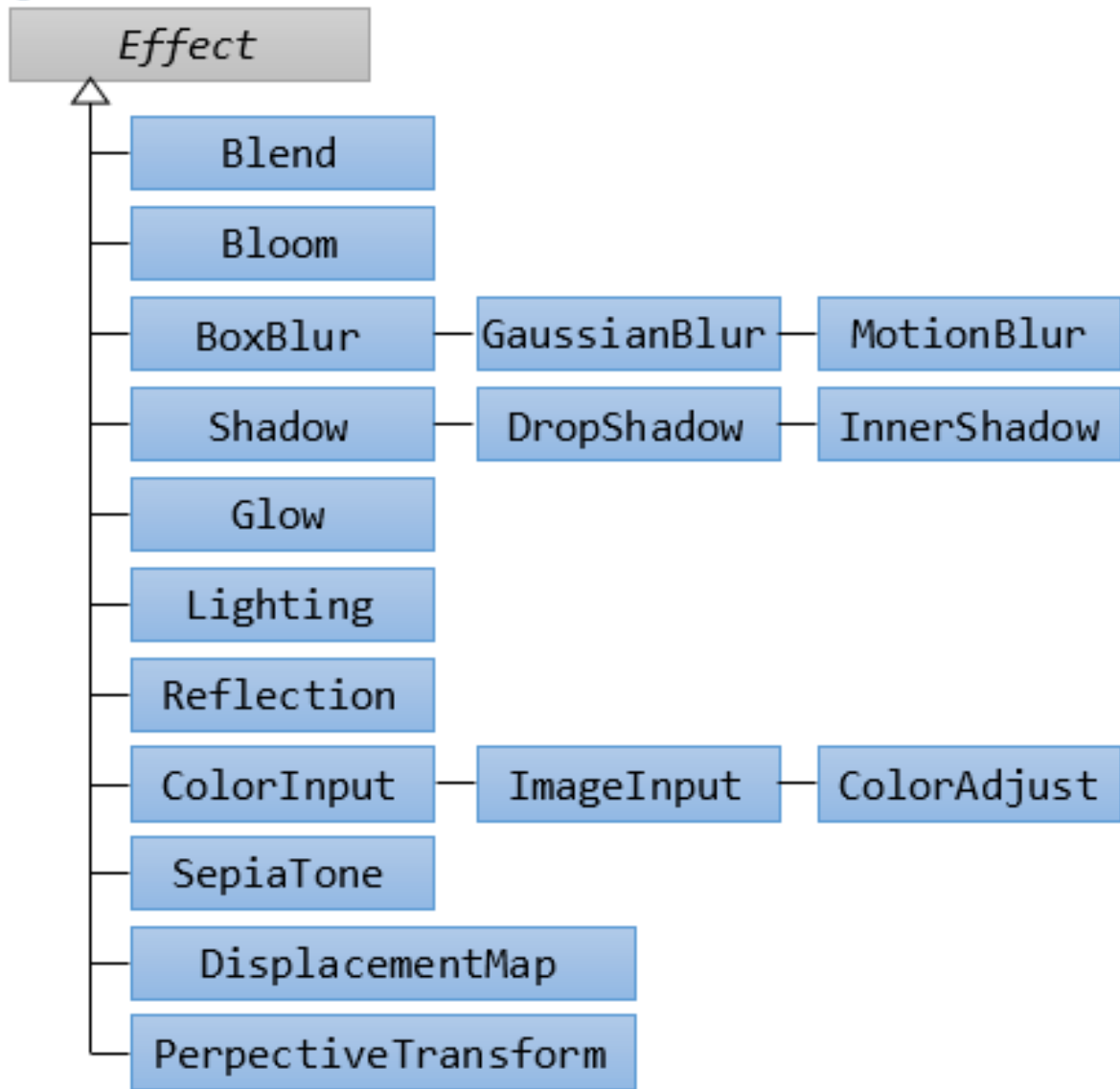
✿ Blend



<https://www.falkhausen.de/JavaFX-10/scene.effect/Effect-examples.html>



`javafx.scene.effect`



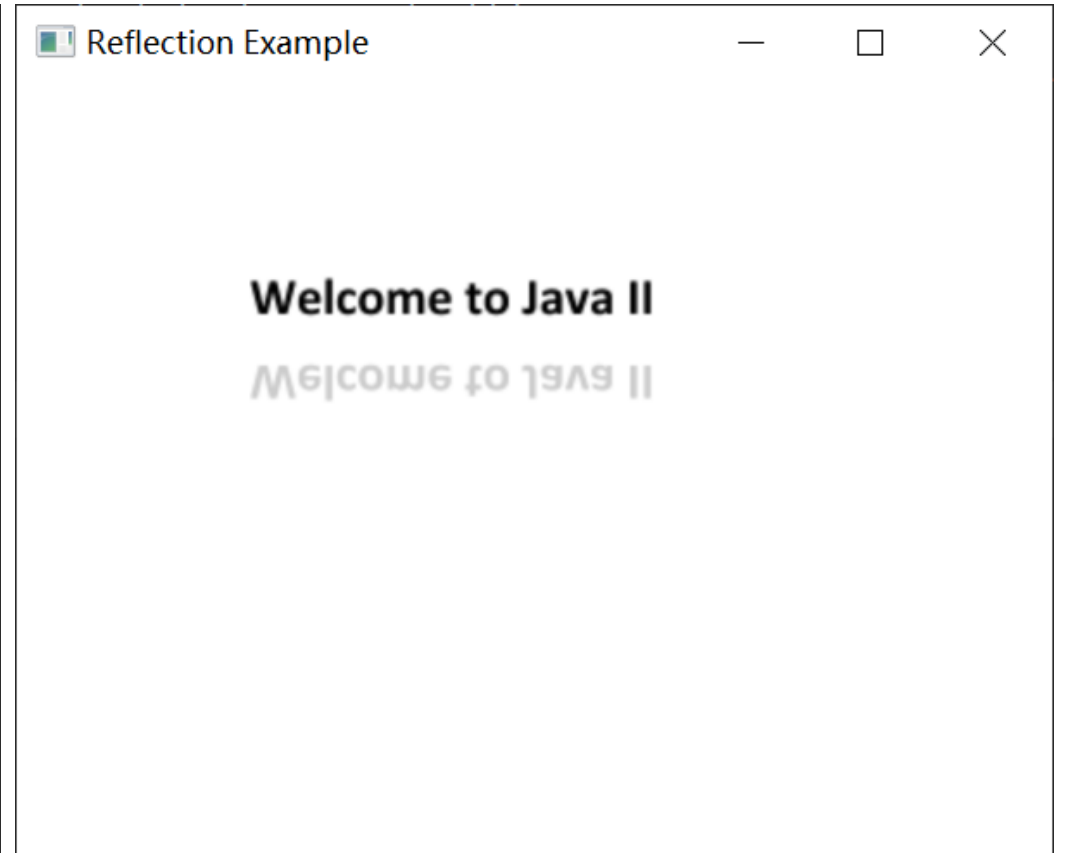
```
Text text = new Text();

Reflection ref = new Reflection();
ref.setBottomOpacity(0.2);
ref.setFraction(12);
ref.setTopOffset(10);
ref.setTopOpacity(0.2);

text.setEffect(ref);

Group root = new Group();
root.getChildren().add(text);

Scene scene = new Scene(root, width: 400, height: 300);
```



Full example: <https://www.javatpoint.com/javafx-reflection-effect>

Example: Reflection Effect

JavaFX Animation

Class Animation

```
java.lang.Object
    javafx.animation.Animation
```

Direct Known Subclasses:

```
Timeline, Transition
```

Class Transition

```
java.lang.Object
    javafx.animation.Animation
        javafx.animation.Transition
```

Direct Known Subclasses:

```
FadeTransition, FillTransition, ParallelTransition, PathTransition,
PauseTransition, RotateTransition, ScaleTransition,
SequentialTransition, StrokeTransition, TranslateTransition
```


JavaFX Animation Example



Creating Path

```
//Creating a Path
Path path = new Path();           Extends Shape

//Moving to the starting point
MoveTo moveTo = new MoveTo( x: 208, y: 71);

//Creating line path to a new point           PathElements
LineTo line1 = new LineTo( x: 421, y: 161); Drawing a straight
LineTo line2 = new LineTo( x: 226, y: 232); line from the current
LineTo line3 = new LineTo( x: 332, y: 52); coordinate to the
LineTo line4 = new LineTo( x: 369, y: 250); new coordinates.
LineTo line5 = new LineTo( x: 208, y: 71);

//Adding all the elements to the path
path.getElements().add(moveTo);
path.getElements().addAll(line1, line2, line3, line4, line5);
```

Full example:

https://www.tutorialspoint.com/javafx/javafx_event_handling.htm

Creating Path Transition Animation

Allows the node to animate through a specified path over the specified duration

```
//Creating the path transition
PathTransition pathTransition = new PathTransition();
//Setting the duration of the transition
pathTransition.setDuration(Duration.millis(1000));
//Setting the node for the transition
pathTransition.setNode(circle);
//Setting the path for the transition
pathTransition.setPath(path);
//Setting the orientation of the path
pathTransition.setOrientation(
    PathTransition.OrientationType.ORTHOGONAL_TO_TANGENT);
//Setting the cycle count for the transition
pathTransition.setCycleCount(50);
//Setting auto reverse value to true
pathTransition.setAutoReverse(false);
```

Full example:

https://www.tutorialspoint.com/javafx/javafx_event_handling.htm

Add the Animation Event

When button is clicked, play the animation

```
Button playButton = new Button( text: "Play");  
playButton.setLayoutX(300);  
playButton.setLayoutY(250);  
playButton.setOnMouseClicked((event -> pathTransition.play()));
```

play() is inherited from the Animation class

Full example:

https://www.tutorialspoint.com/javafx/javafx_event_handling.htm



Lecture 9

- **JavaFX**

- Overview
- Hello World
- Design & Concepts
- Layouts, Shapes, UI controls
- Charts and Axis
- Transformation, Animation, Effects
- **FXML**
- **Multithreading in JavaFX**

JavaFX FXML

- **Motivation**

- Design code (appearance) are mixed with the application code (event handling logics)
- Code will be easier to maintain if application design is separated from the application logic

- **JavaFX FXML**

- An XML-based language
- Allows users to build the user interface separate from the application logic



FXML Structure

A .fxml file to
design the user
interface

A controller class
to implement the
application logic

Bootstrap JavaFX Application

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.VBox?>

<?import javafx.scene.control.Button?>
<VBox alignment="CENTER" spacing="20.0" xmlns:fx="http://javafx.com/fxml"
    fx:controller="com.example.cs209a_lectures_javafx.HelloController">
    <padding>
        <Insets bottom="20.0" left="20.0" right="20.0" top="20.0"/>
    </padding>

    <Label fx:id="welcomeText"/>
    <Button text="Hello!" onAction="#onHelloButtonClick"/>
</VBox>
```

hello-view.fxml

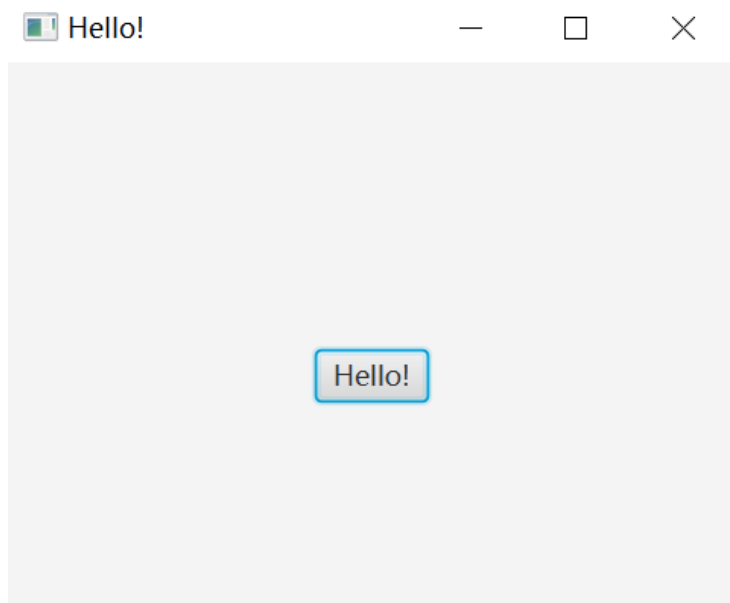
```
import javafx.fxml.FXML;
import javafx.scene.control.Label;

public class HelloController {
    @FXML
    private Label welcomeText;

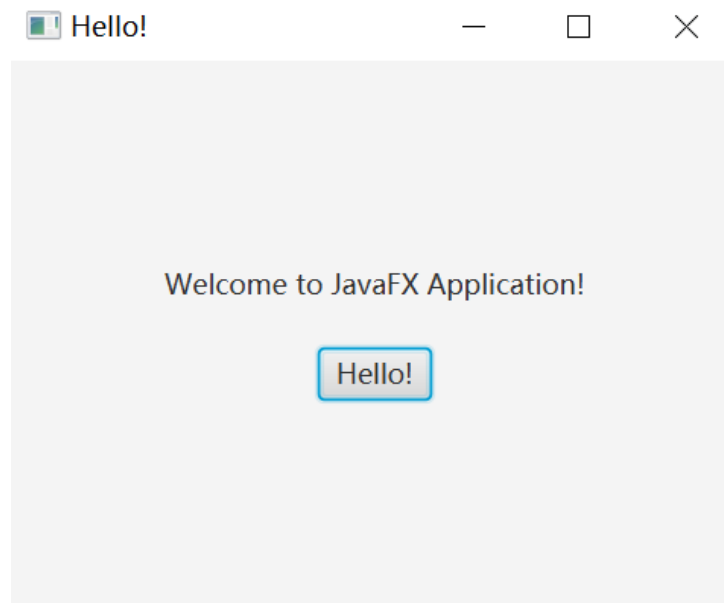
    @FXML
    protected void onHelloButtonClick() {
        welcomeText.setText("Welcome to JavaFX Application!");
    }
}
```

HelloController.java

Bootstrap JavaFX Application



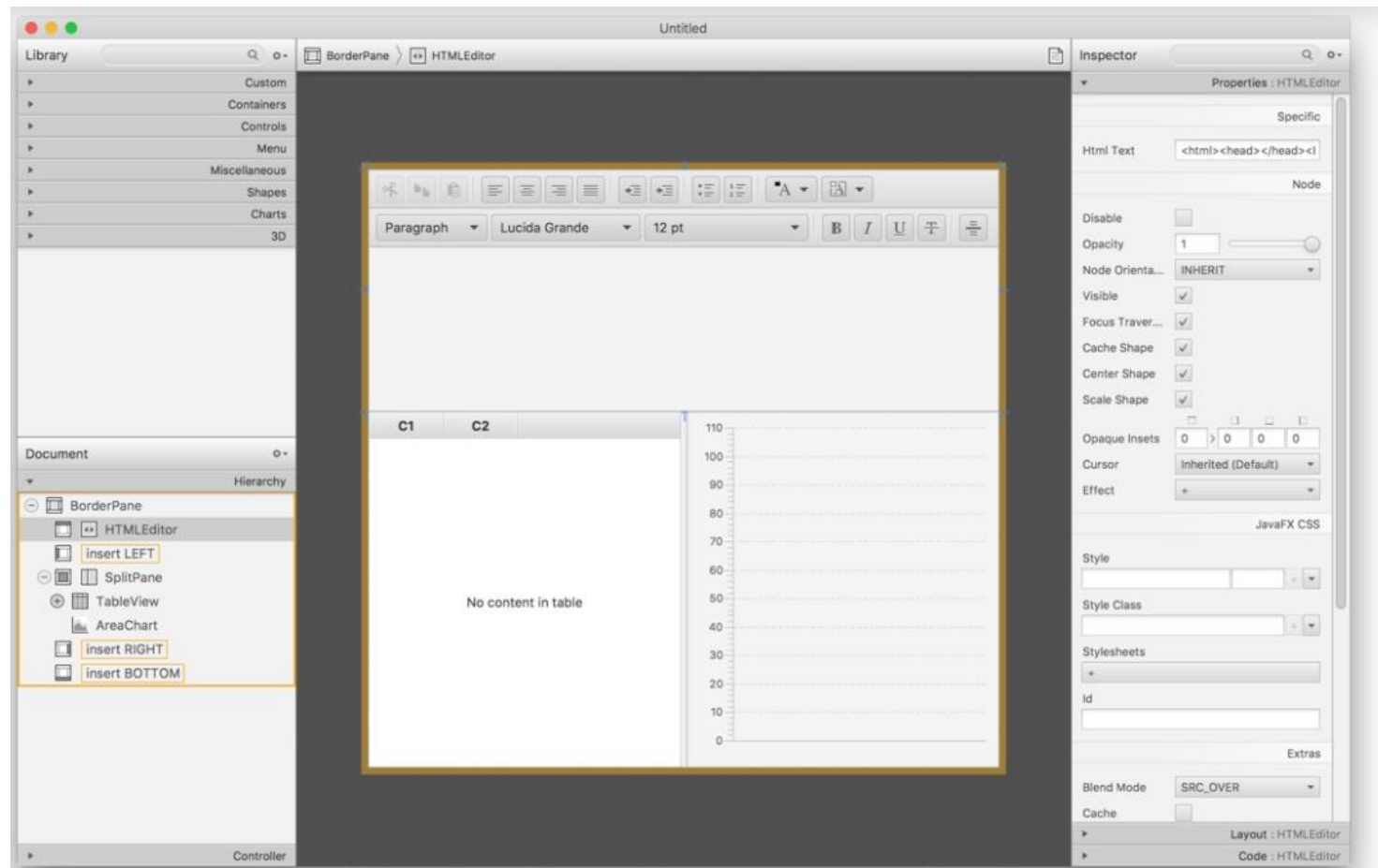
On start



Button clicked

JavaFX Scene Builder

A visual layout tool that lets users quickly design JavaFX application user interfaces by drag and drop



Drag & Drop,
Rapid Application
Development.

[Download Now](#)

JavaFX Threading

- JavaFX UI can only be accessed and modified from the **JavaFX Application thread**
- Creation of JavaFX Scene and Stage objects as well as modification of scene graph must be done on the **JavaFX application thread**.
- Executing long-running tasks within the JavaFX application thread (e.g., in `start()`) make the GUI unresponsive until the task is completed.
- No GUI controls react to mouse clicks, mouse over, keyboard input while the JavaFX application thread is busy running that task.

JavaFX Threading

- A best practice is to do long-running tasks on one or more background threads
- The background thread informs the JavaFX Application thread whenever it is time to update the UI.
- JavaFX Platform.runLater() takes a Runnable which will be executed by the JavaFX application thread when it has time. From inside this Runnable you can modify the JavaFX scene graph.

```
Thread taskThread = new Thread(new Runnable() {
    @Override
    public void run() {
        double progress = 0;
        for(int i=0; i<10; i++){

            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            progress += 0.1;
            final double reportedProgress = progress;

            Platform.runLater(new Runnable() {
                @Override
                public void run() {
                    progressBar
                        .setProgress(reportedProgress);
                }
            });
        }
    }
});

taskThread.start();
```

Reference: <https://jenkov.com/tutorials/javafx/concurrency.html>

Next Lecture

- Reflections
- Annotations