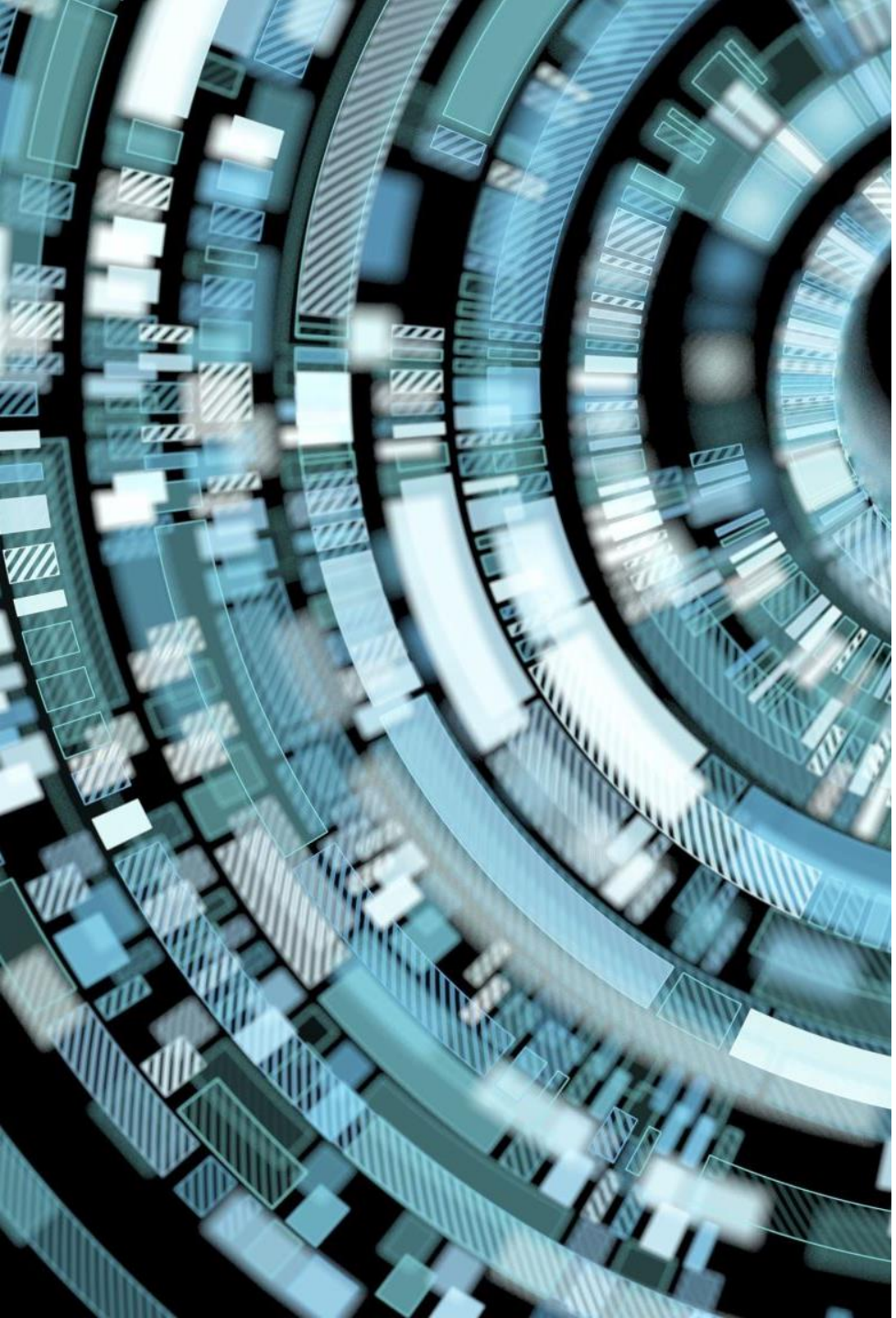
Computer System Design & Application

计算机系统设计与应用A

陶伊达 （TAO Yida）
taoyd@sustech.edu.cn

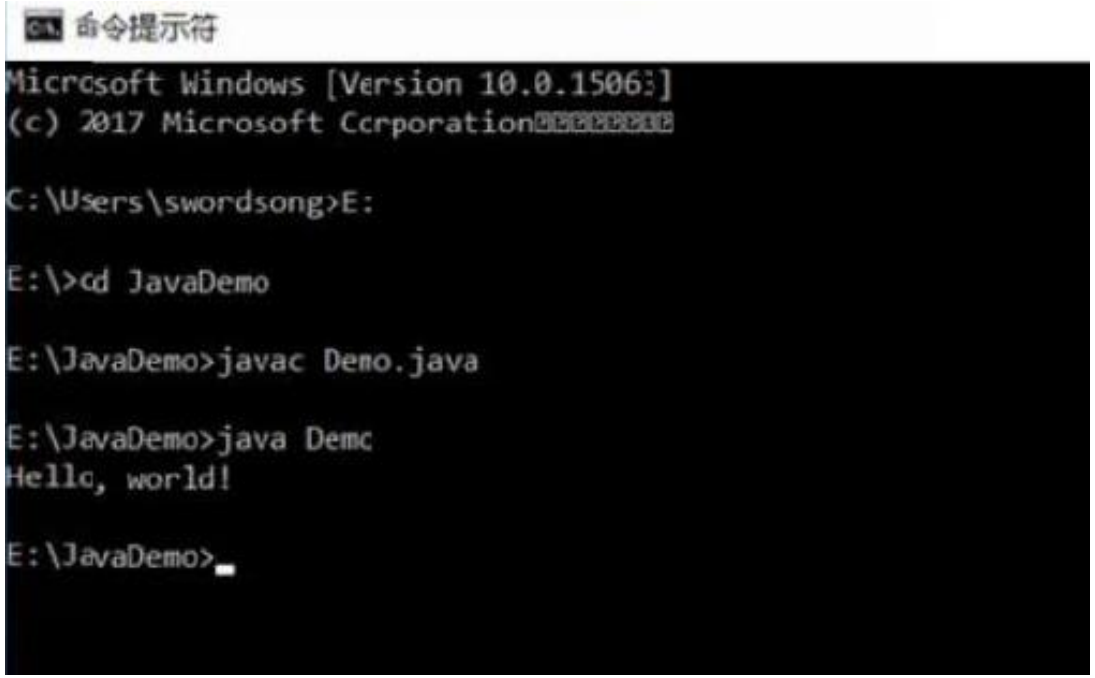


Lecture 9

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

GUI Overview

- **G**raphical **U**ser **I**nterface (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)



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

JavaFX Overview

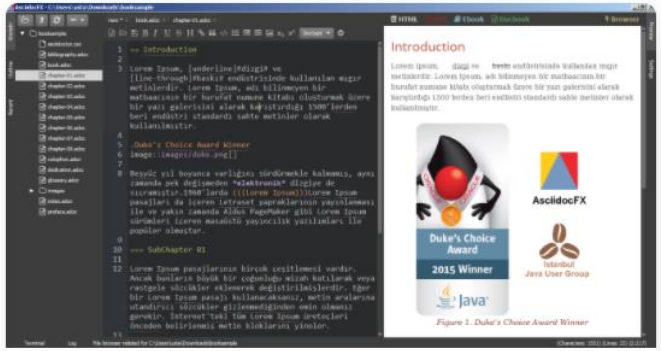
https://openjfx.io/

- 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

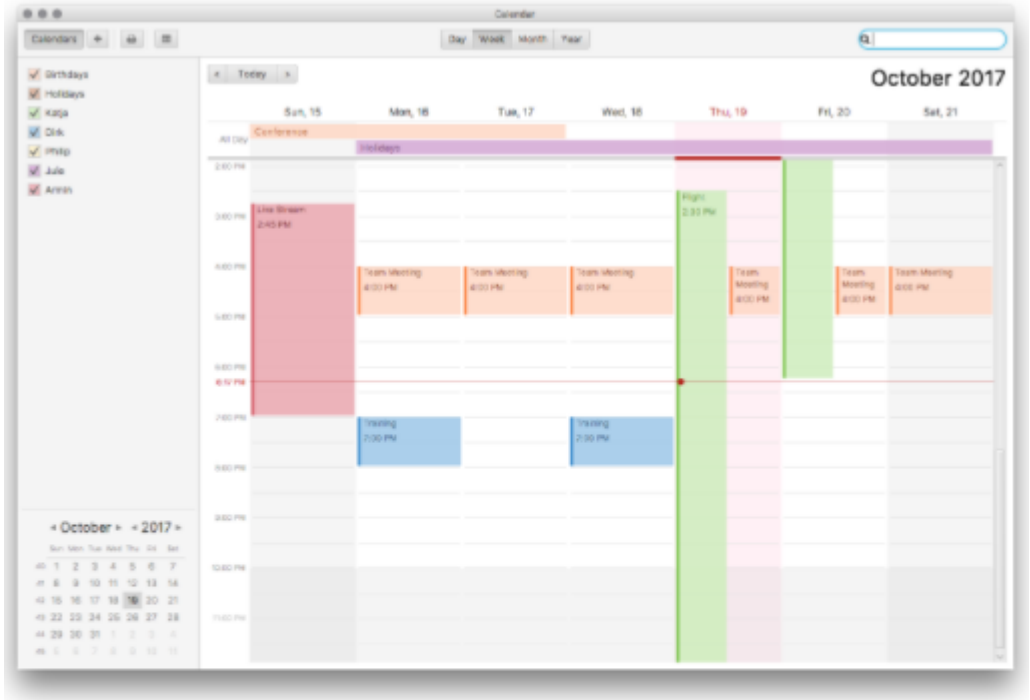
JavaFX Showcases

Images from JavaFX official site



AsciidocFX

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



CalendarFX

A Java framework for creating sophisticated calendar views

Gluon Maps

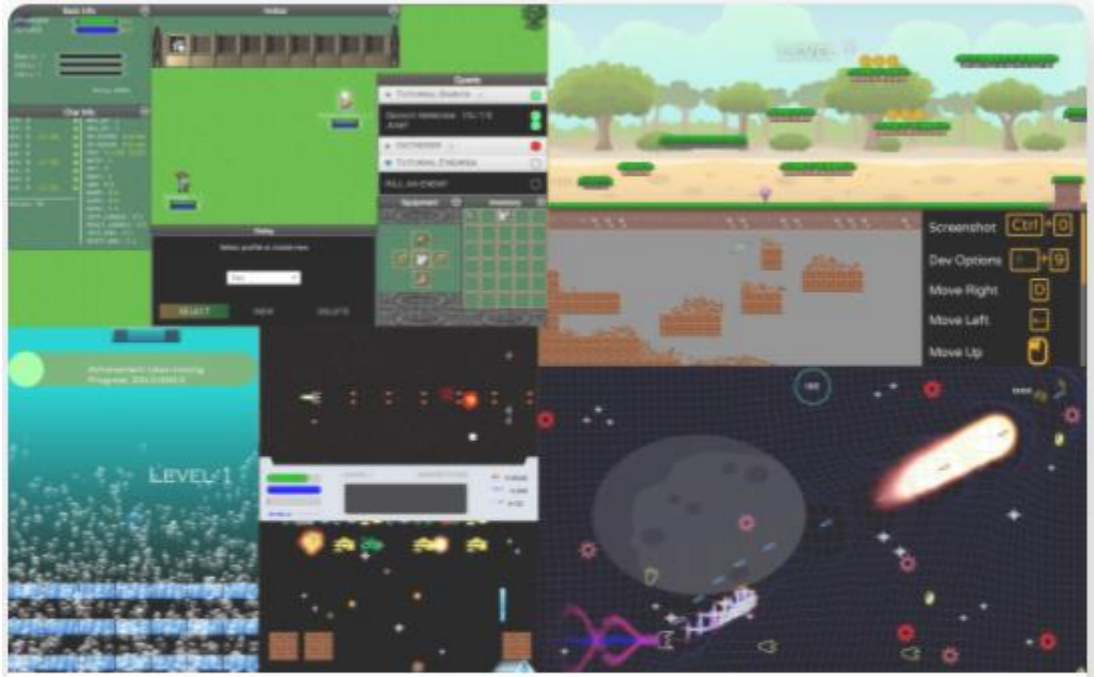
Tiles based geo-location map framework

JavaFX Showcases

Images from JavaFX official site

TilesFX

A JavaFX library containing tiles for Dashboards



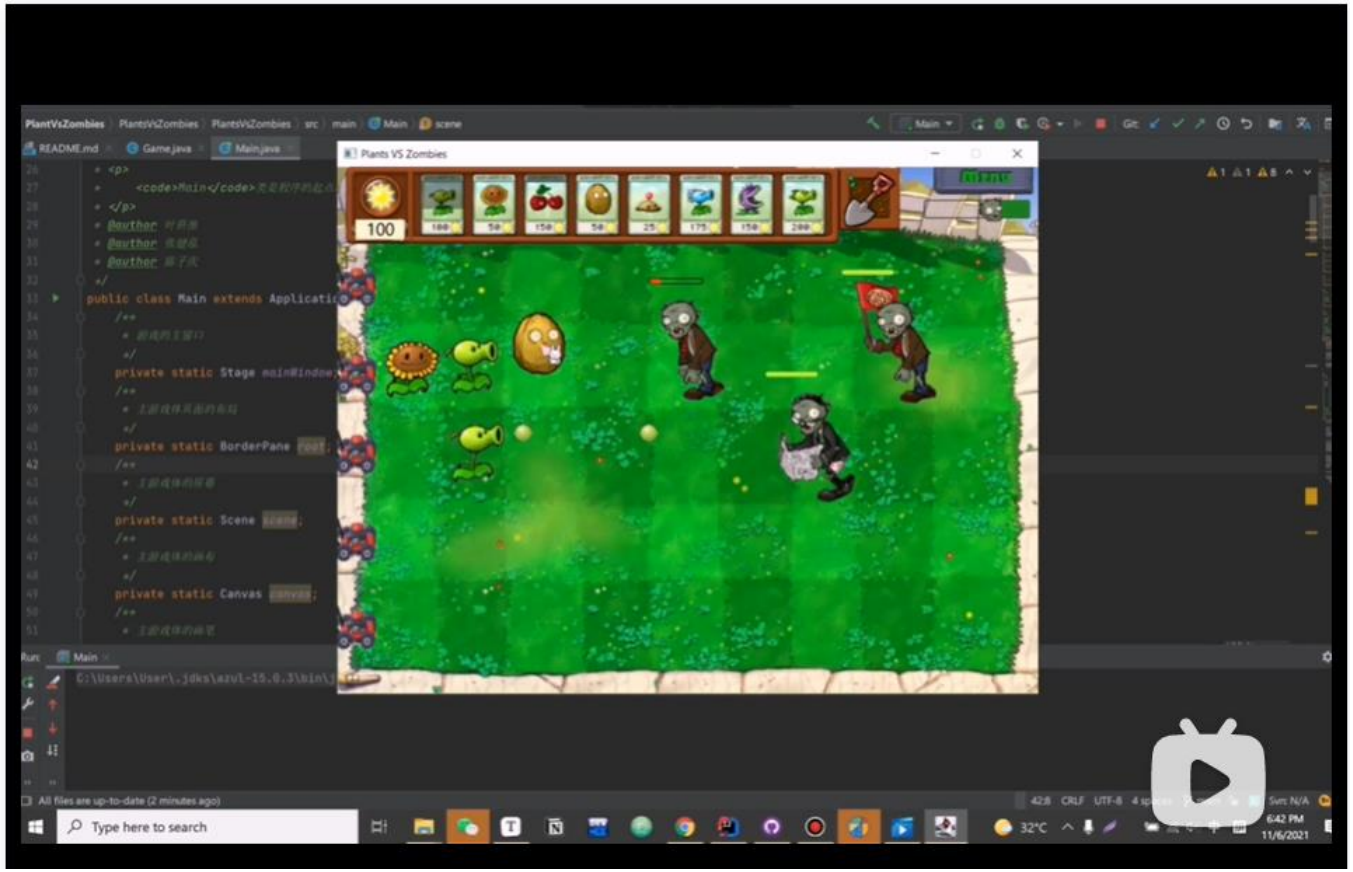
FXGL

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

```java
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

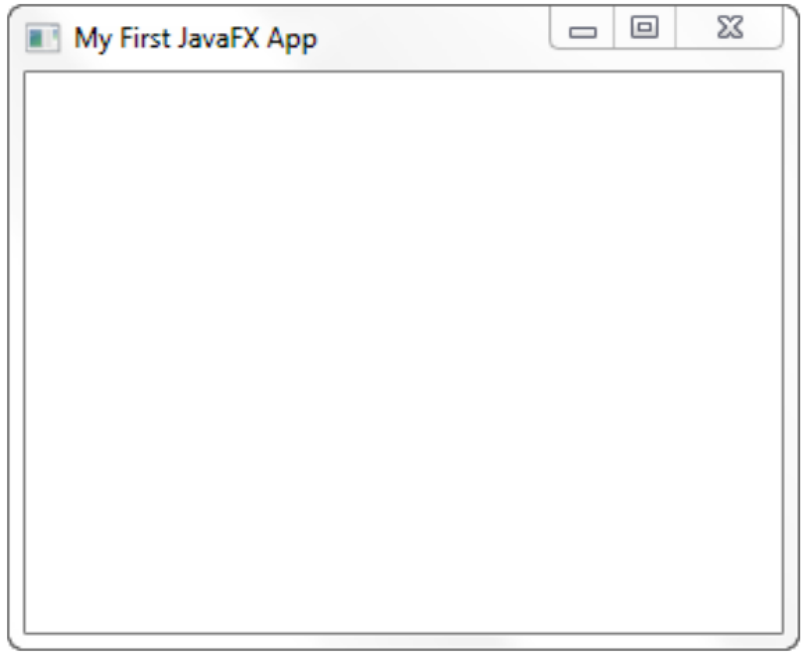
Implement the abstract `start()` method of the `Application` class (called when a JavaFX application starts)

`launch()` launches the JavaFX application.

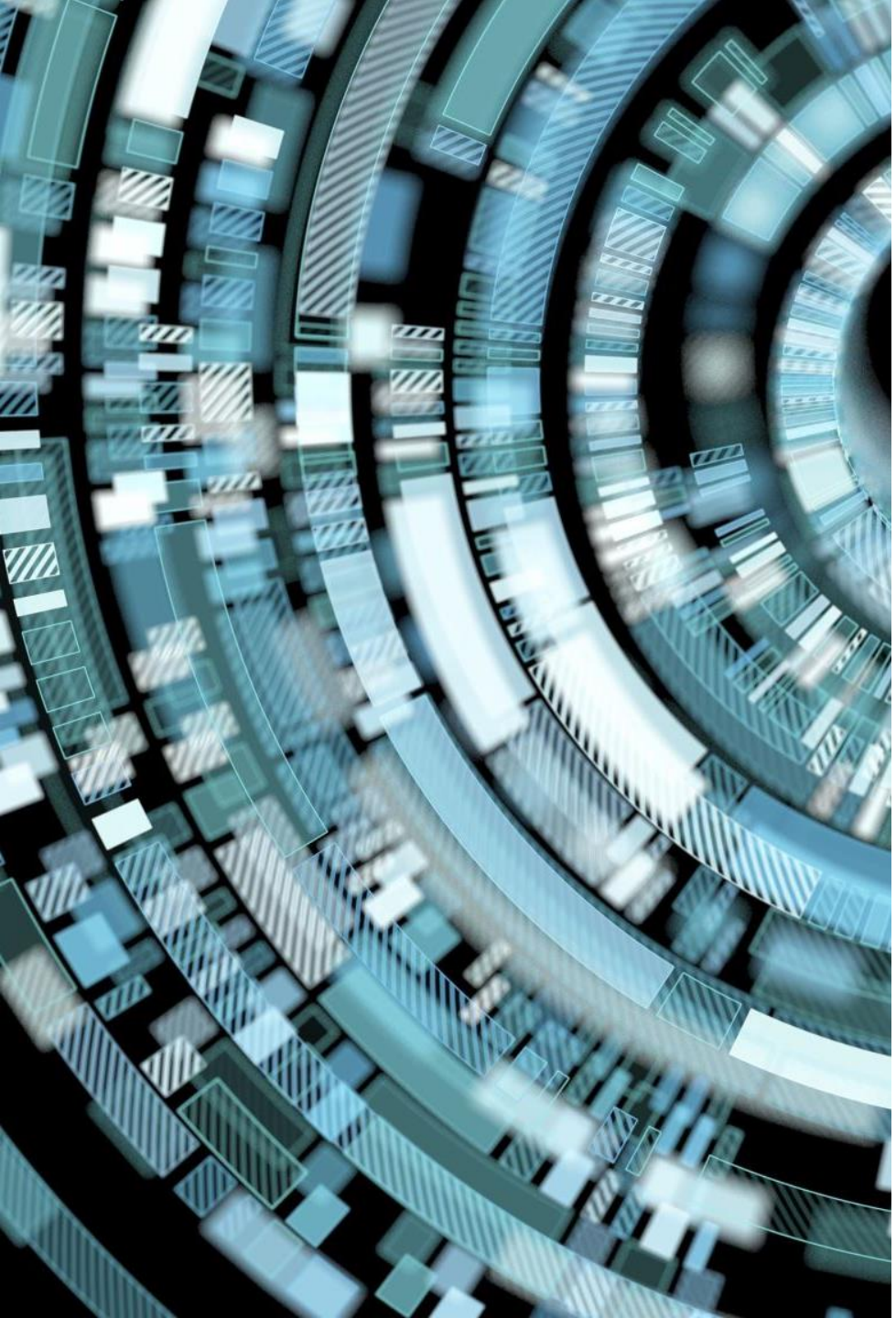
http://tutorials.jenkov.com/javafx/your-first-javafx-application.html

JavaFX Hello World

```java
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);
    }


}
```



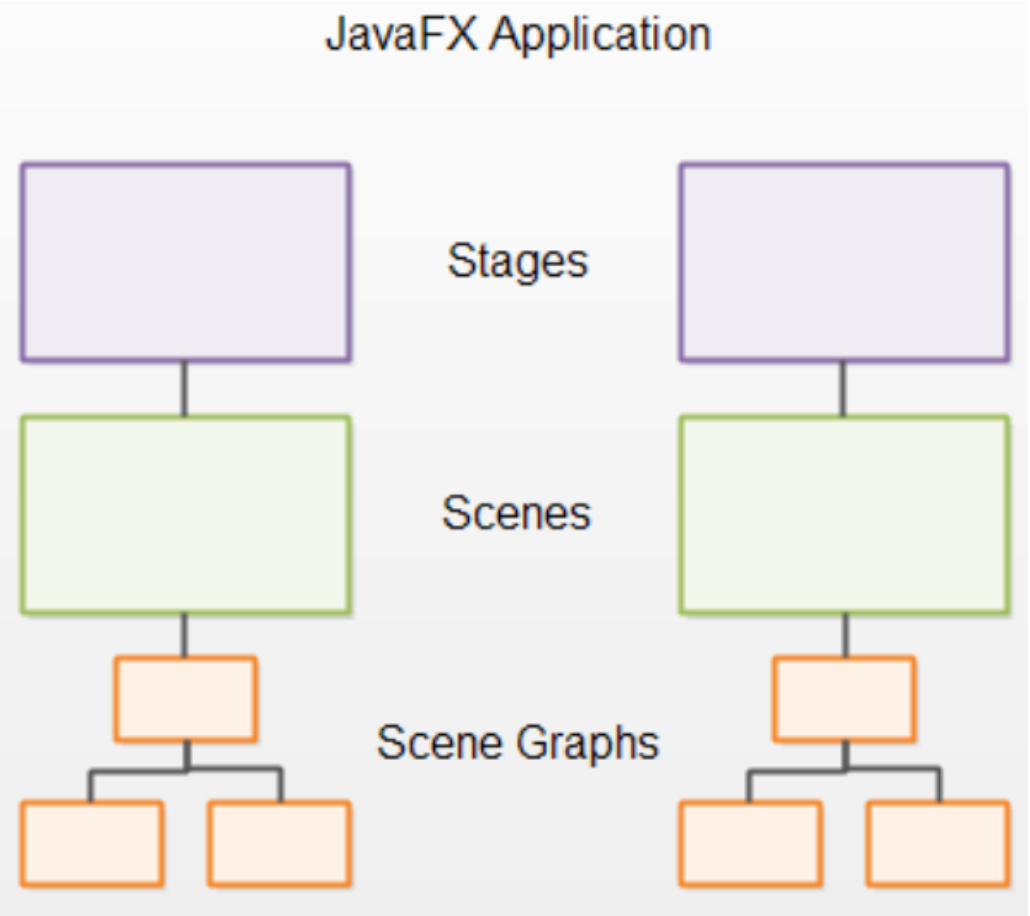
http://tutorials.jenkov.com/javafx/your-first-javafx-application.html



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

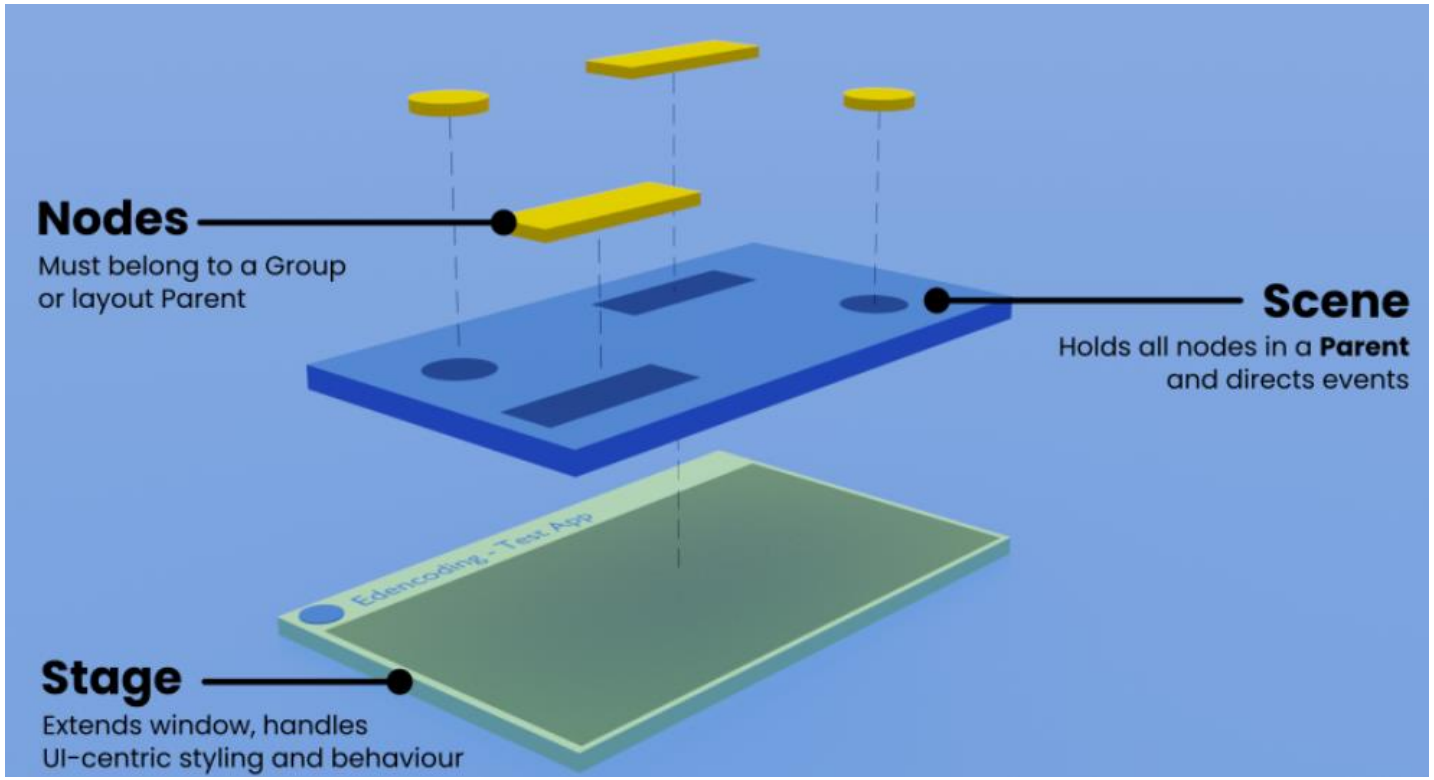


Image source: https://edencoding.com/javafx-scene/

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

```java
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

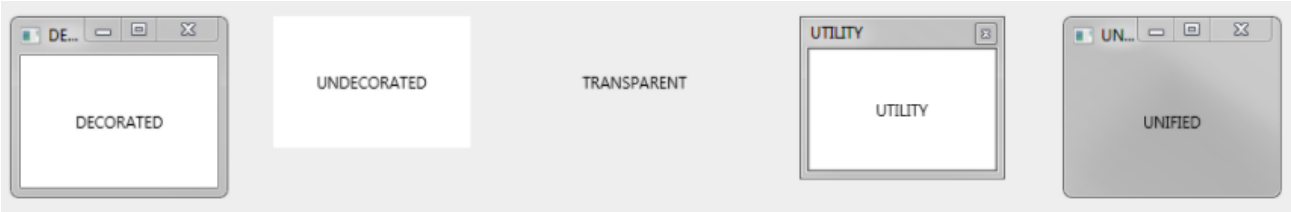
Please refer to the official documentation for full details

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, undec
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

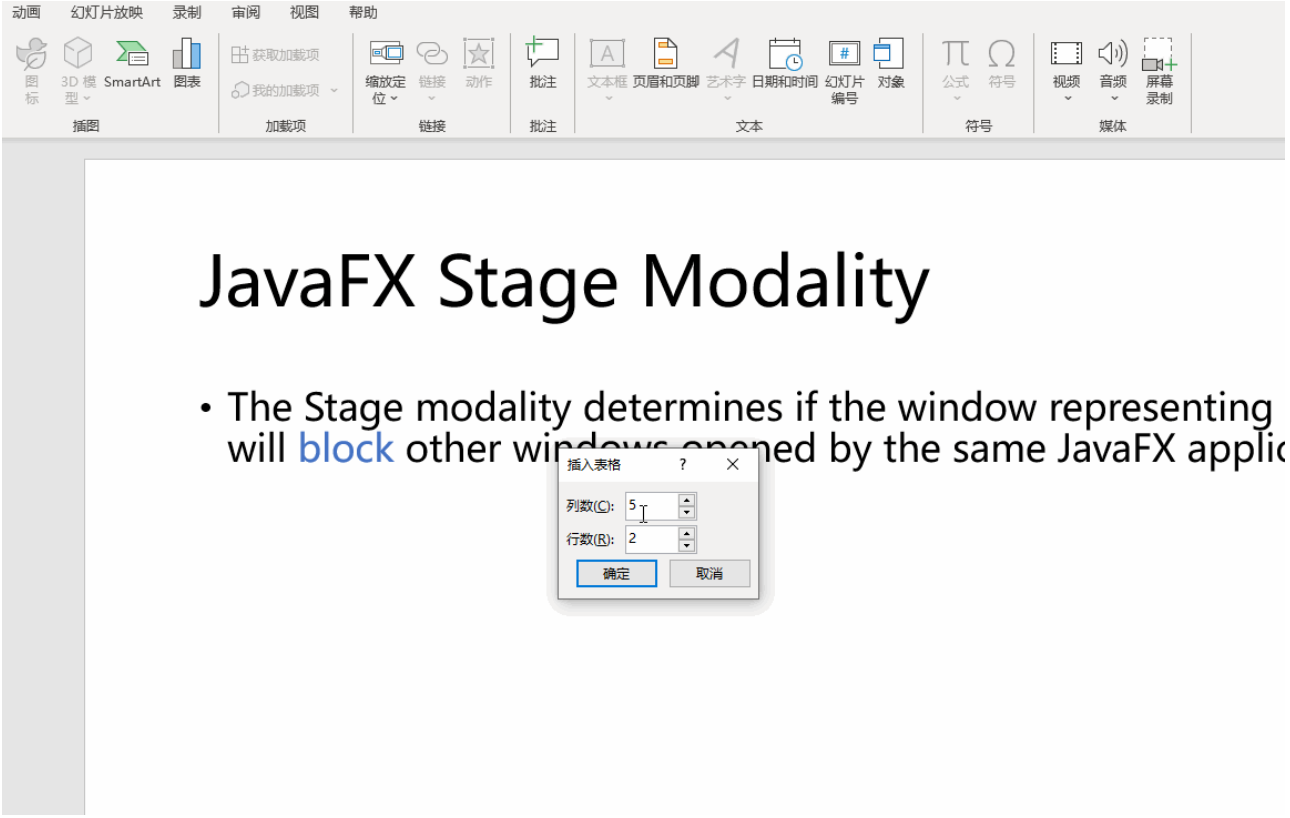
java.lang.Object
    java.lang.Enum<StageStyle>
        javafx.stage.StageStyle

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.



JavaFX Stage Modality

Enum Modality

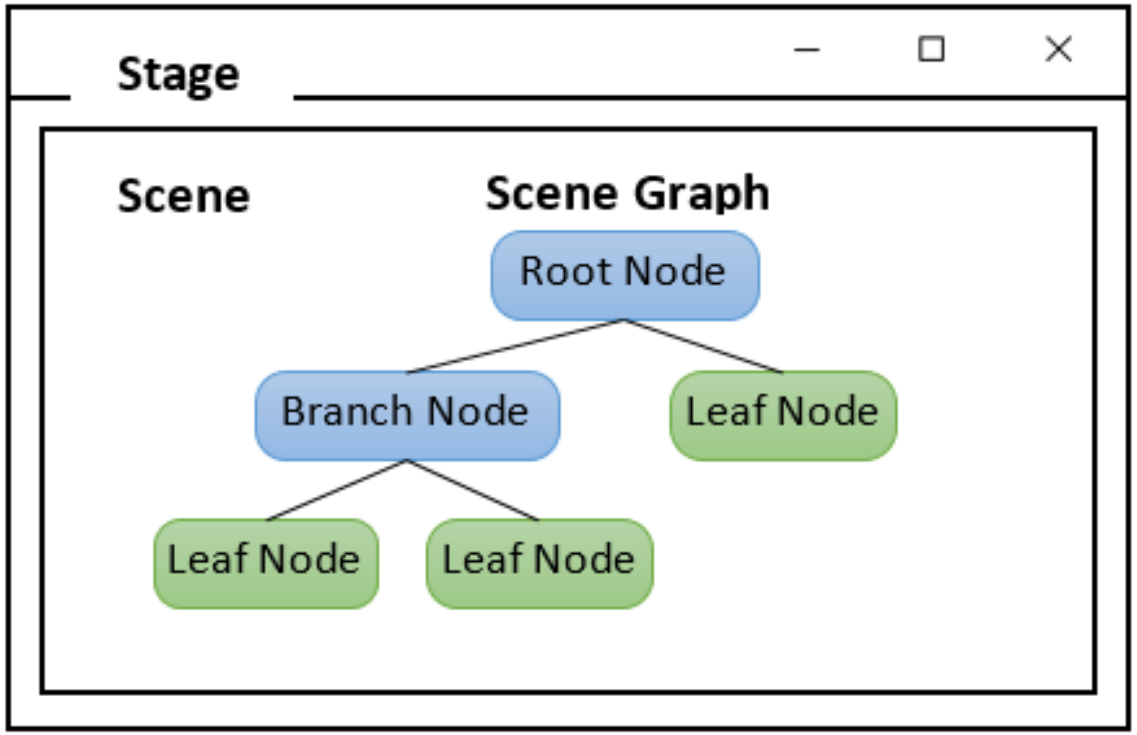
java.lang.Object
    java.lang.Enum<Modality>
        javafx.stage.Modality

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.

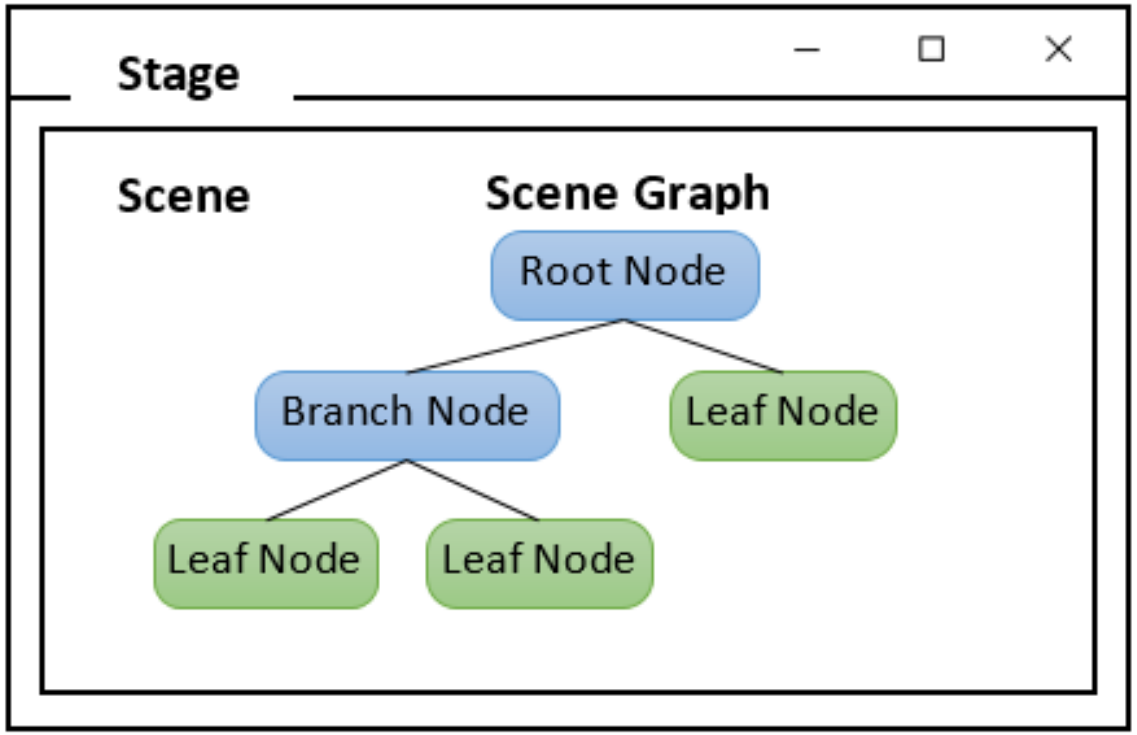
JavaFX Scene

- 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.

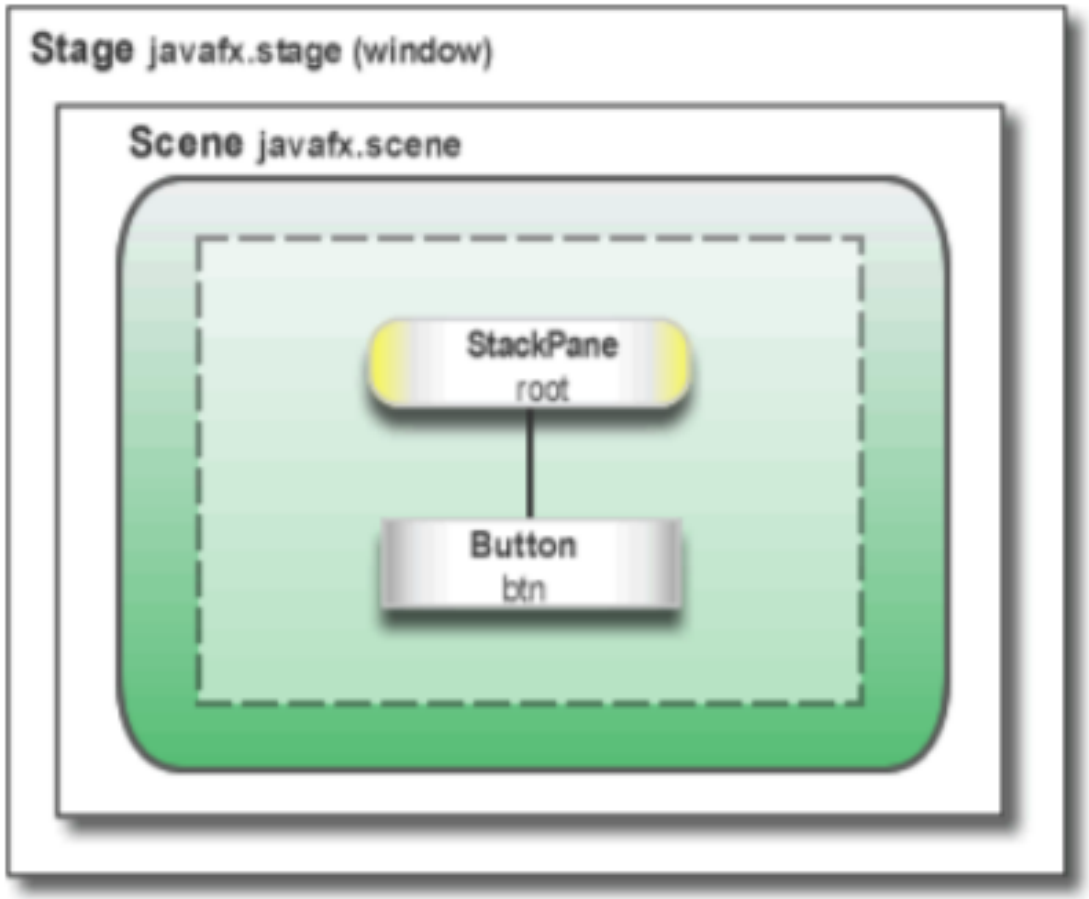


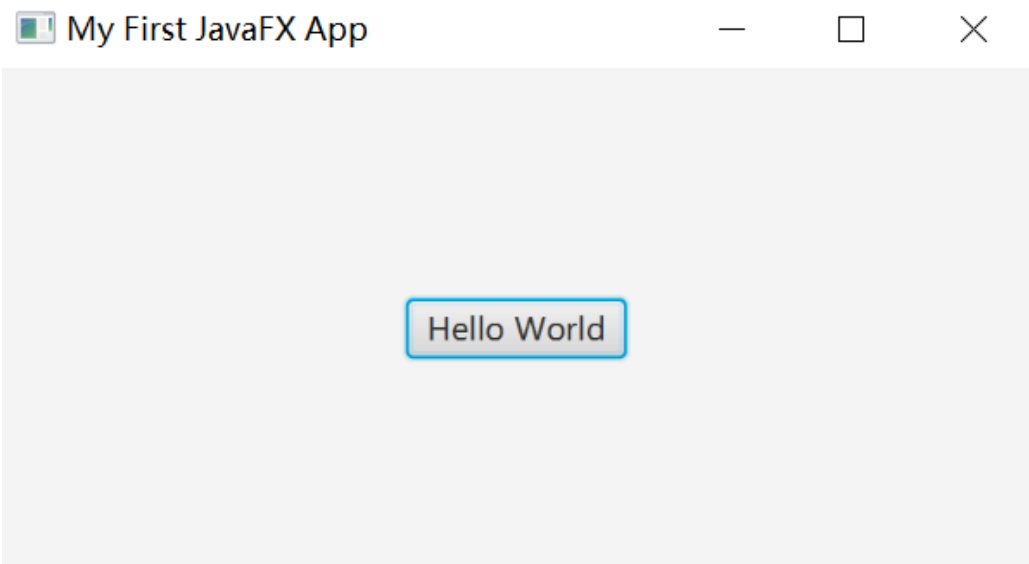
Scene Graph

- 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





```java
@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

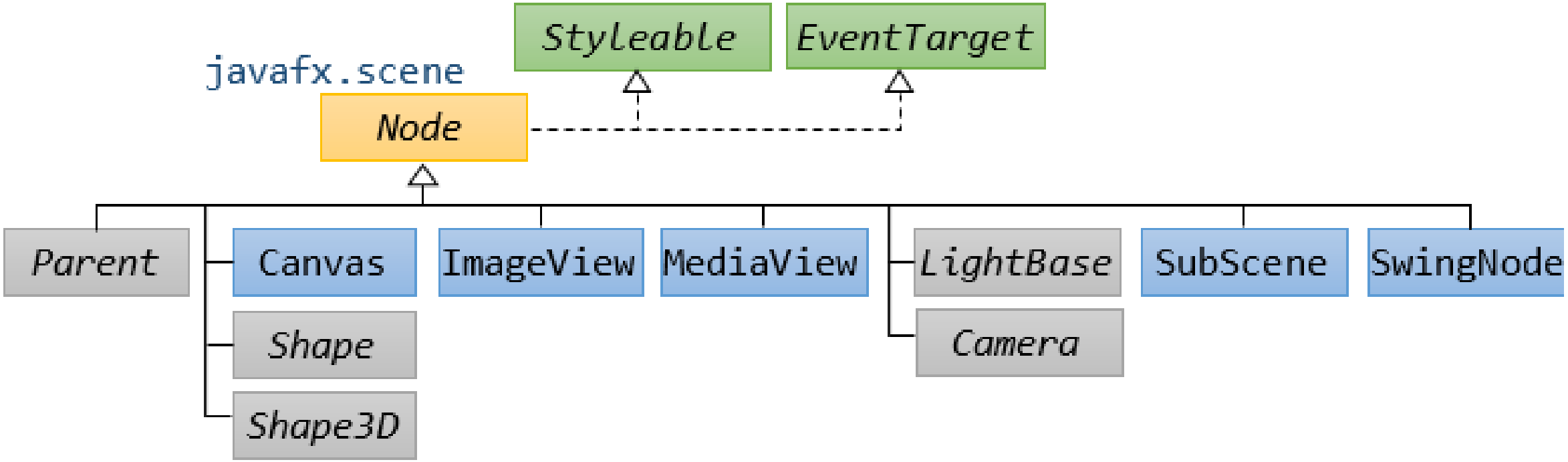
2. The child node is a Button object, with an event handler for printing a message when pressed

3. Add button to the root node
4. Create a scene with the root
5. Set the scene for the stage and show

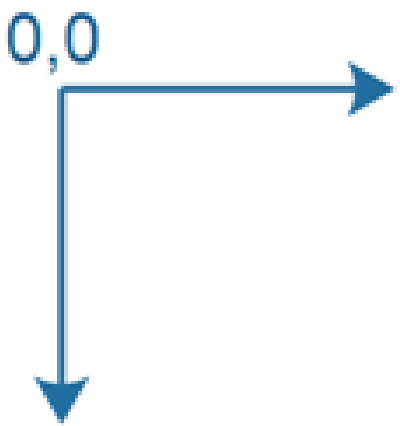
Node

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

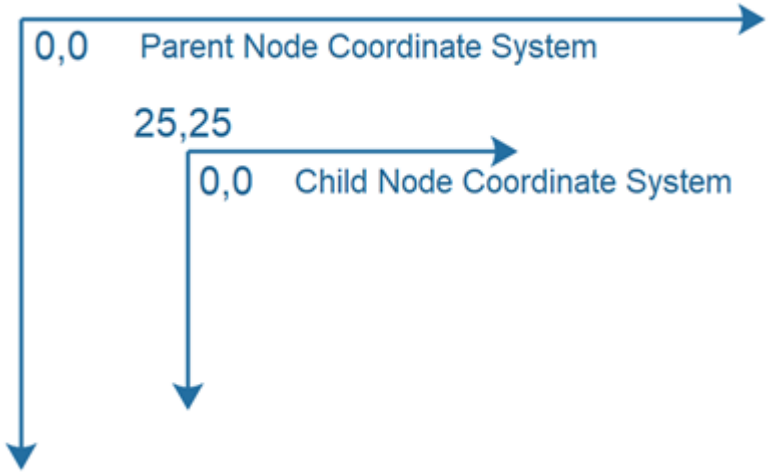
https://www3.ntu.edu.sg/home/ehchua/programming/java/Javafx1_intro.html



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.

ReadOnlyBooleanProperty	focused Indicates whether this Node currently has
BooleanProperty	focusTraversable Specifies whether this Node should be a pa
ReadOnlyBooleanProperty	hover Whether or not this Node is being hovered
StringProperty	id The id of this Node.
ObjectProperty<InputMethodRequests>	inputMethodRequests Property holding InputMethodRequests.
ReadOnlyObjectProperty<Bounds>	layoutBounds The rectangular bounds that should be use
DoubleProperty	layoutX Defines the x coordinate of the translation
DoubleProperty	layoutY Defines the y coordinate of the translation

DoubleProperty	opacity Specifies how opaque (that is, solid) the Node appears.
ReadOnlyObjectProperty<Parent>	parent The parent of this Node.
BooleanProperty	pickOnBounds Defines how the picking computation is done for this node when
ReadOnlyBooleanProperty	pressed Whether or not the Node is pressed.
DoubleProperty	rotate Defines the angle of rotation about the Node's center, measured
ObjectProperty<Point3D>	rotationAxis Defines the axis of rotation of this Node.
DoubleProperty	scaleX Defines the factor by which coordinates are scaled about the cen
DoubleProperty	scaleY Defines the factor by which coordinates are scaled about the cen
DoubleProperty	scaleZ

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)
```

setOnMouseClicked

MouseEvent

Functional Interface

onKeyPressed
Defines a function to be called

onKeyReleased
Defines a function to be called

onKeyTyped
Defines a function to be called

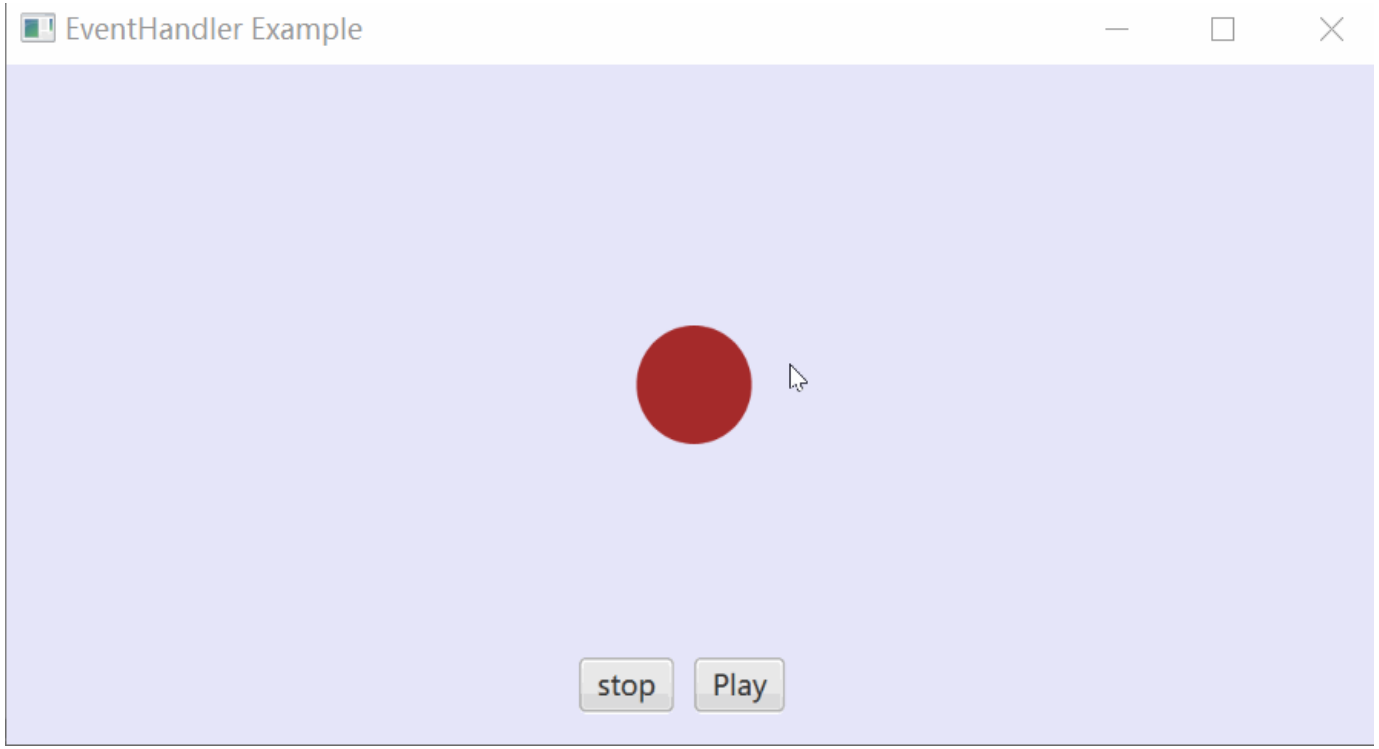
onMouseClicked
Defines a function to be called

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?



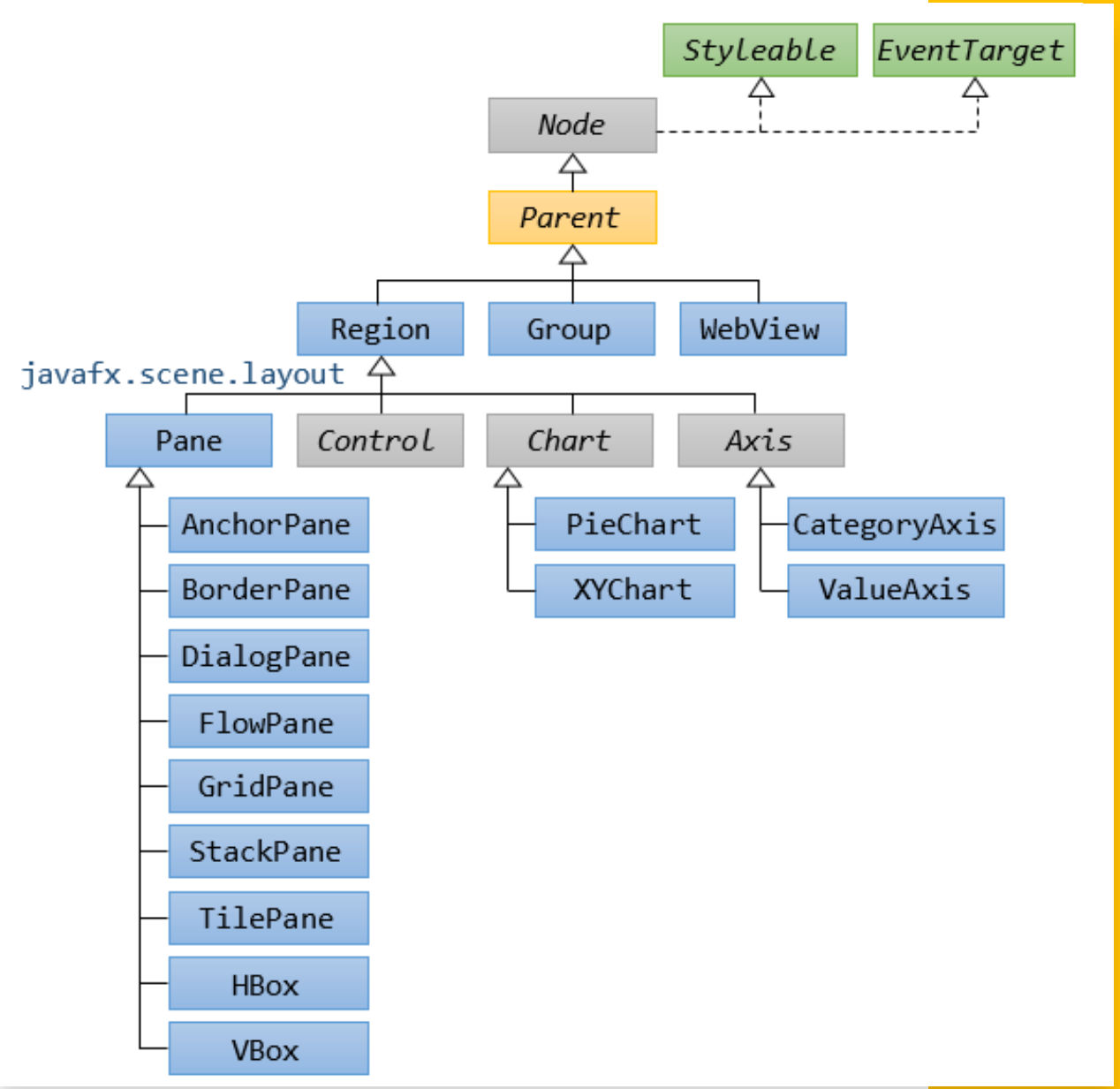

```java
circle.setOnMouseClicked (new EventHandler<MouseEvent>() {
    @Override
    public void handle(javafx.scene.input.MouseEvent e) {
        circle.setFill(Color.DARKSLATEBLUE);
    }
});
playButton.setOnMouseClicked((new EventHandler<MouseEvent>() {
    public void handle(MouseEvent event) {
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

```java
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 ...

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

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

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

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

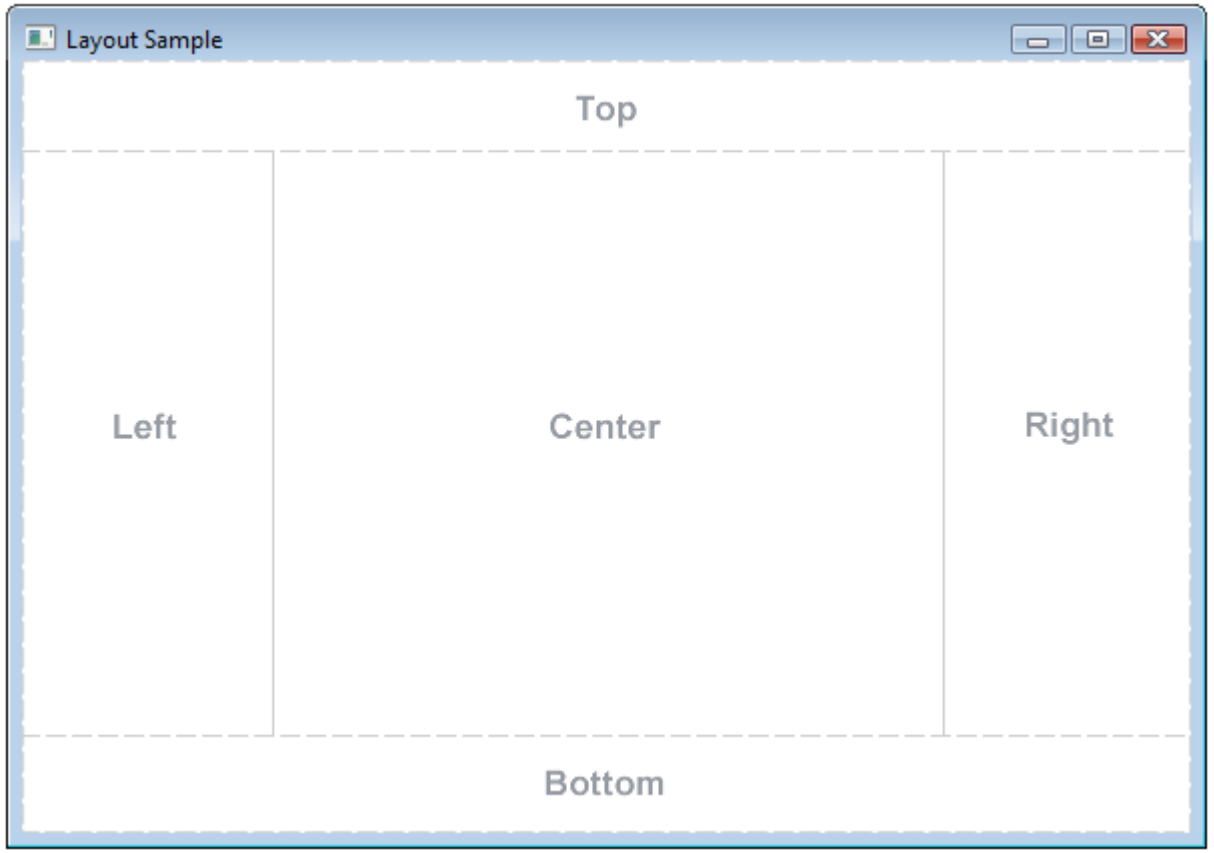
VBOX
VBox (JavaFX 8) - Oracle. children - The initial set of children for this pane. Since: ...

TilePane
TilePane (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.

Figure 1-1 Sample Border Pane

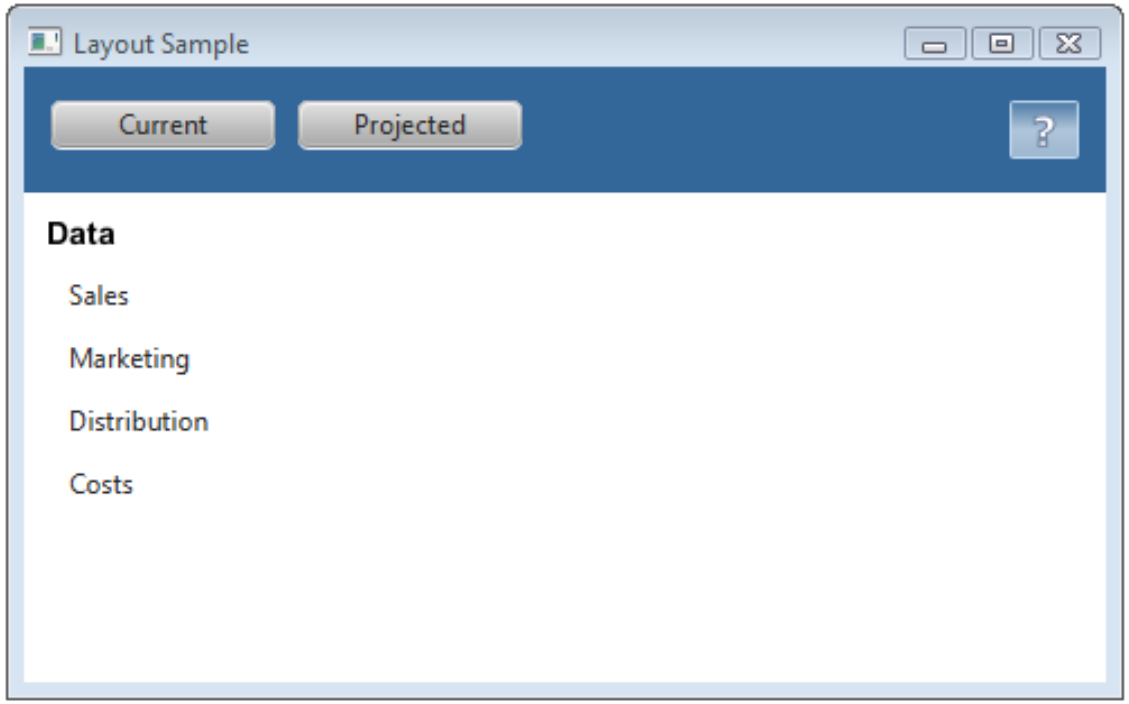


For more details:
https://docs.oracle.com/javafx/2/layout/builtin_layouts.htm

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

Figure 1-5 VBox Pane in a Border Pane

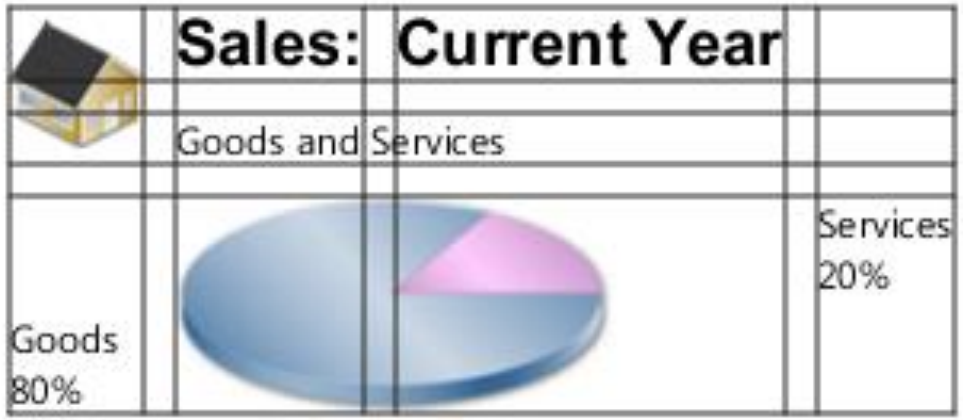


For more details:
https://docs.oracle.com/javafx/2/layout/builtin_layouts.htm

GridPane

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

Figure 1-8 Sample Grid Pane



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

Combine Panes

Different Panes can be combined to make beautiful layout

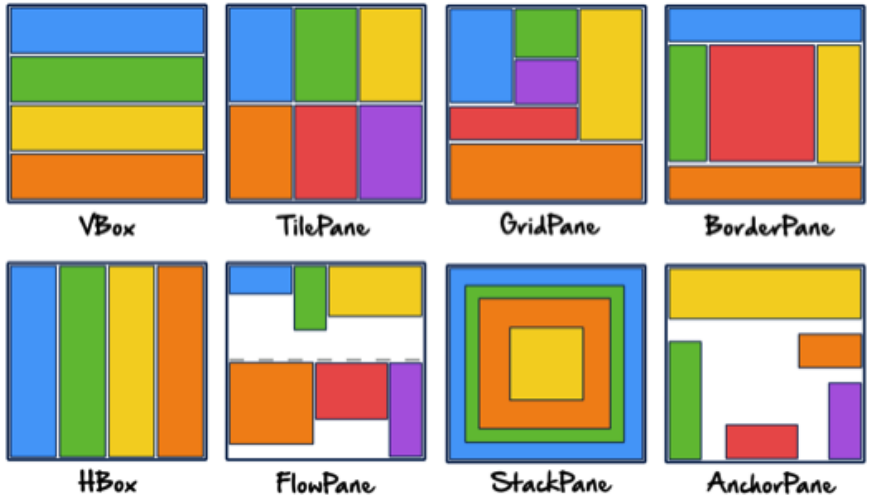
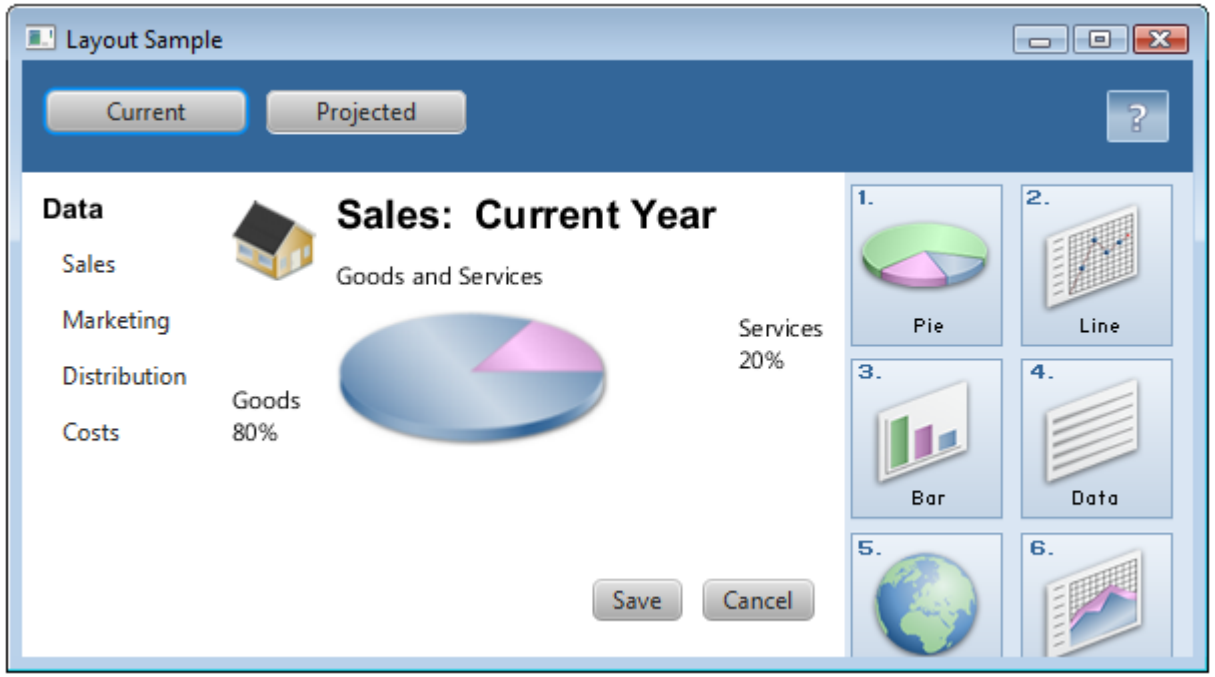


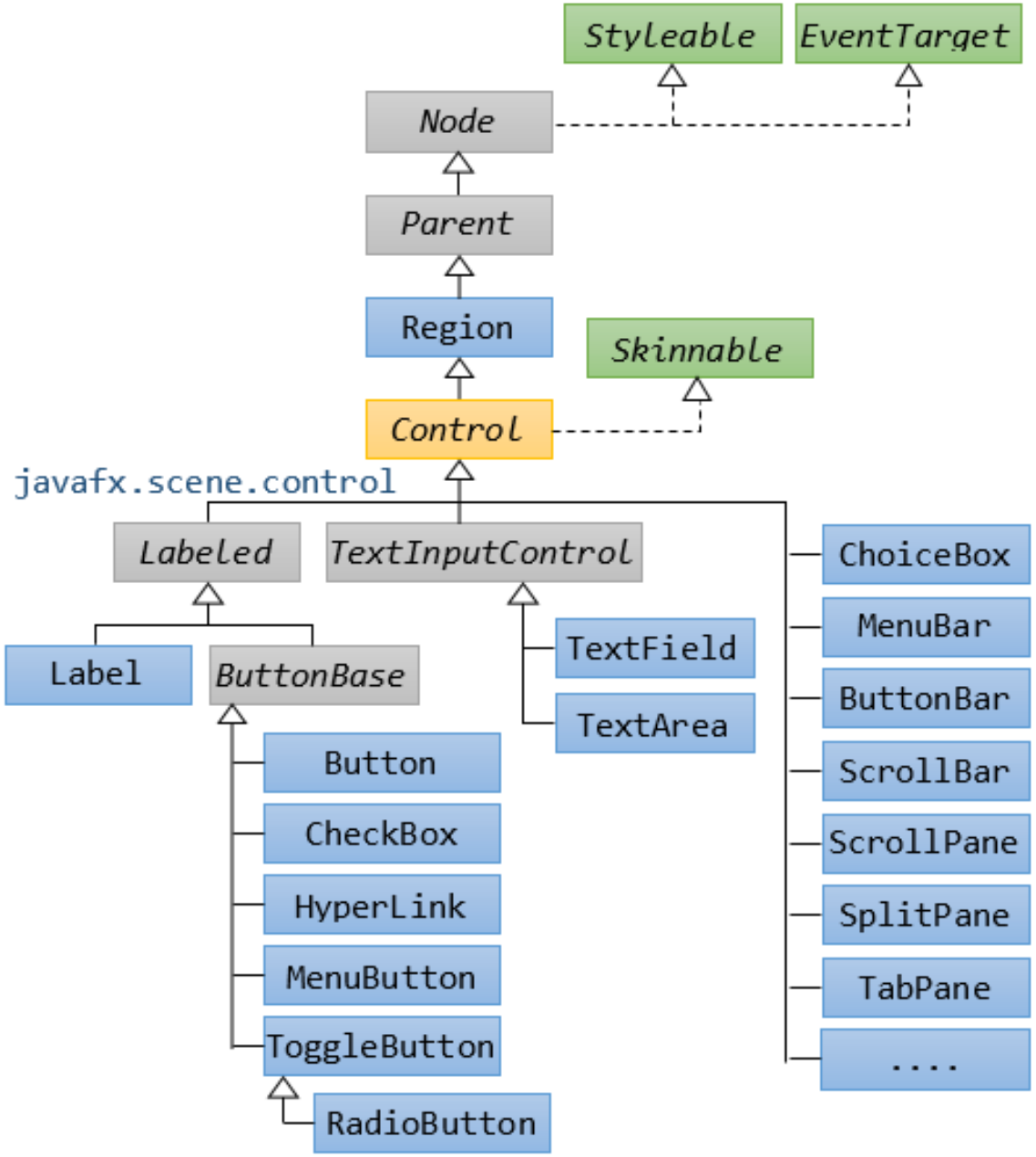
Image source: https://dzone.com/refcardz/javafx-8-1



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

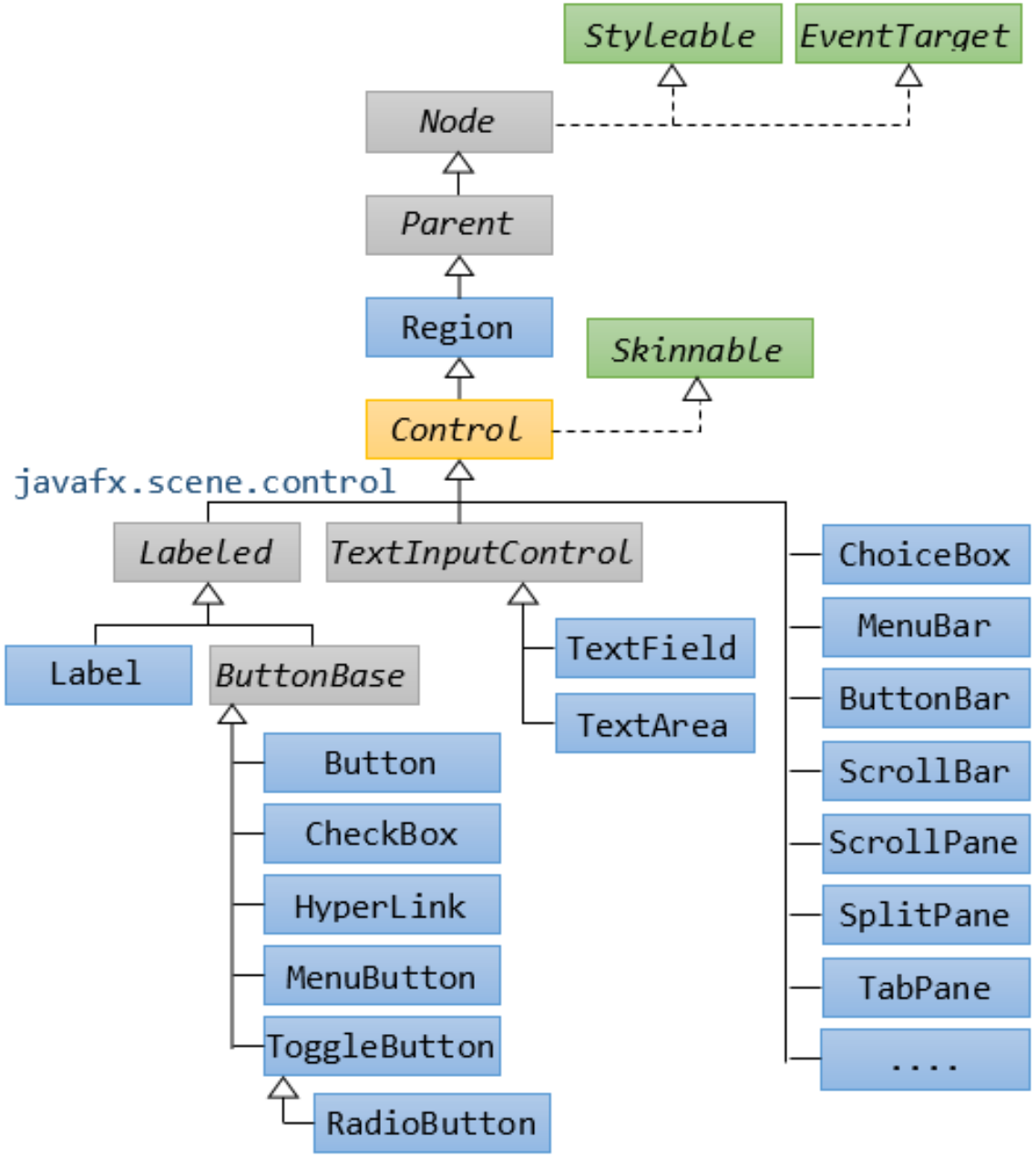
UI Controls

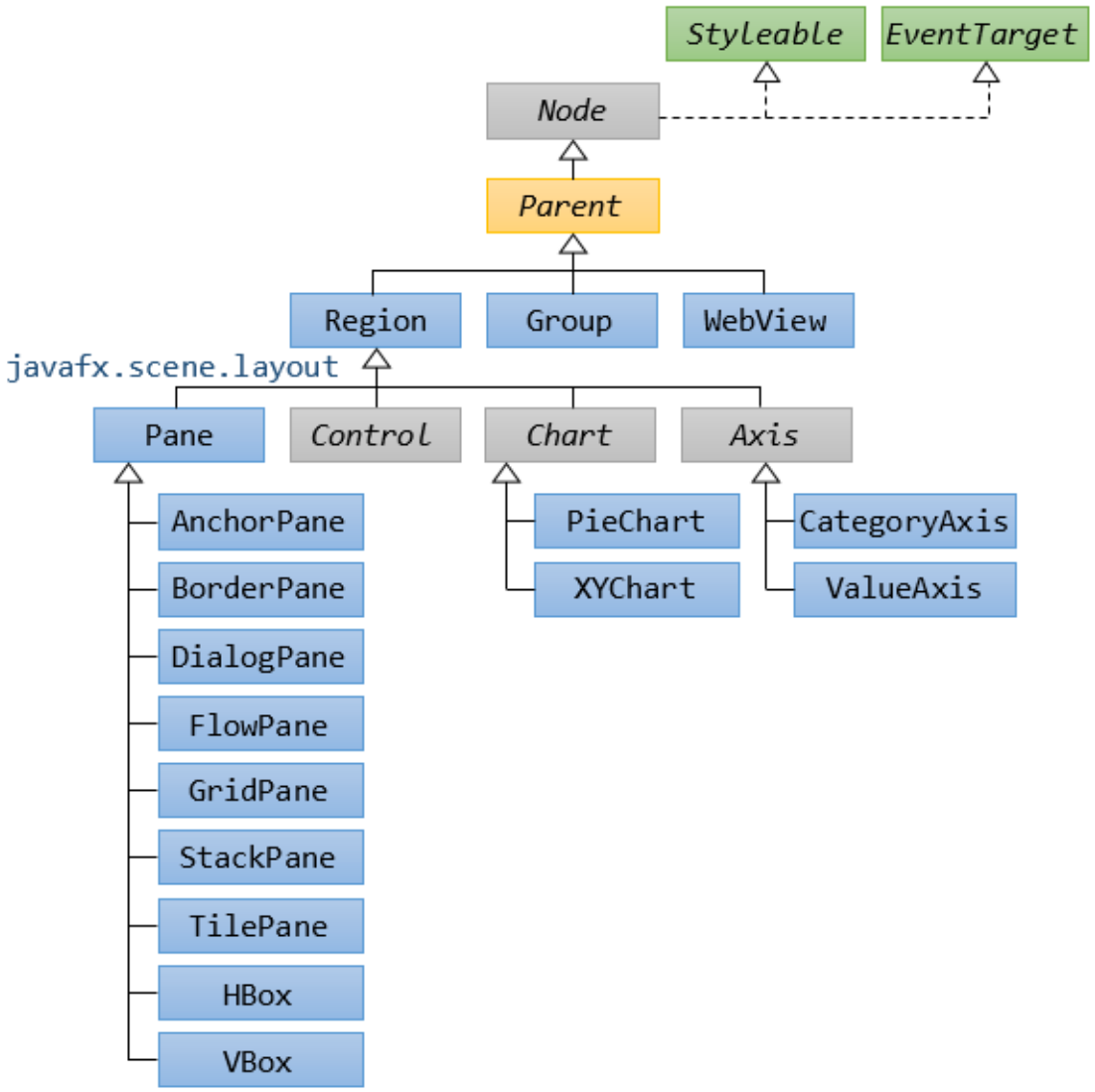

Node 1
Node 2
String
Node 3
Accordion
Check Boxes
Yellow
Orange
Blue
Color Button
Button
Graphic Button
Hyperlink
Hyperlink
Radio Buttons
Cat
Dog
Horse
Toggle Button
Options
Choice Box
Row 1
Row 2
Horizontal List View
Row 1
Row 2
Long Row 3
Simple List View
Progress Bar
60%
Done
Progress Indicator
Scroll Bar
First
Jacob
Isabella
Ethan
Table
Tab 1
Tab 2
Tab
Graphic Label
Advanced Label
Simple Label
Simple Label
Some text
Text Field
Options
Tool Bar
Root node
Child Node 1
Child Node 2
Child Node 3
Tree View


Styleable
EventTarget
Node
Parent
Region
Skinnable
Control
javafx.scene.control
Labeled
TextInputControl
Label
ButtonBase
TextField
TextArea
Button
CheckBox
HyperLink
MenuButton
ToggleButton
RadioButton
ChoiceBox
MenuBar
ButtonBar
ScrollBar
ScrollPane
SplitPane
TabPane
....

UI Controls

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





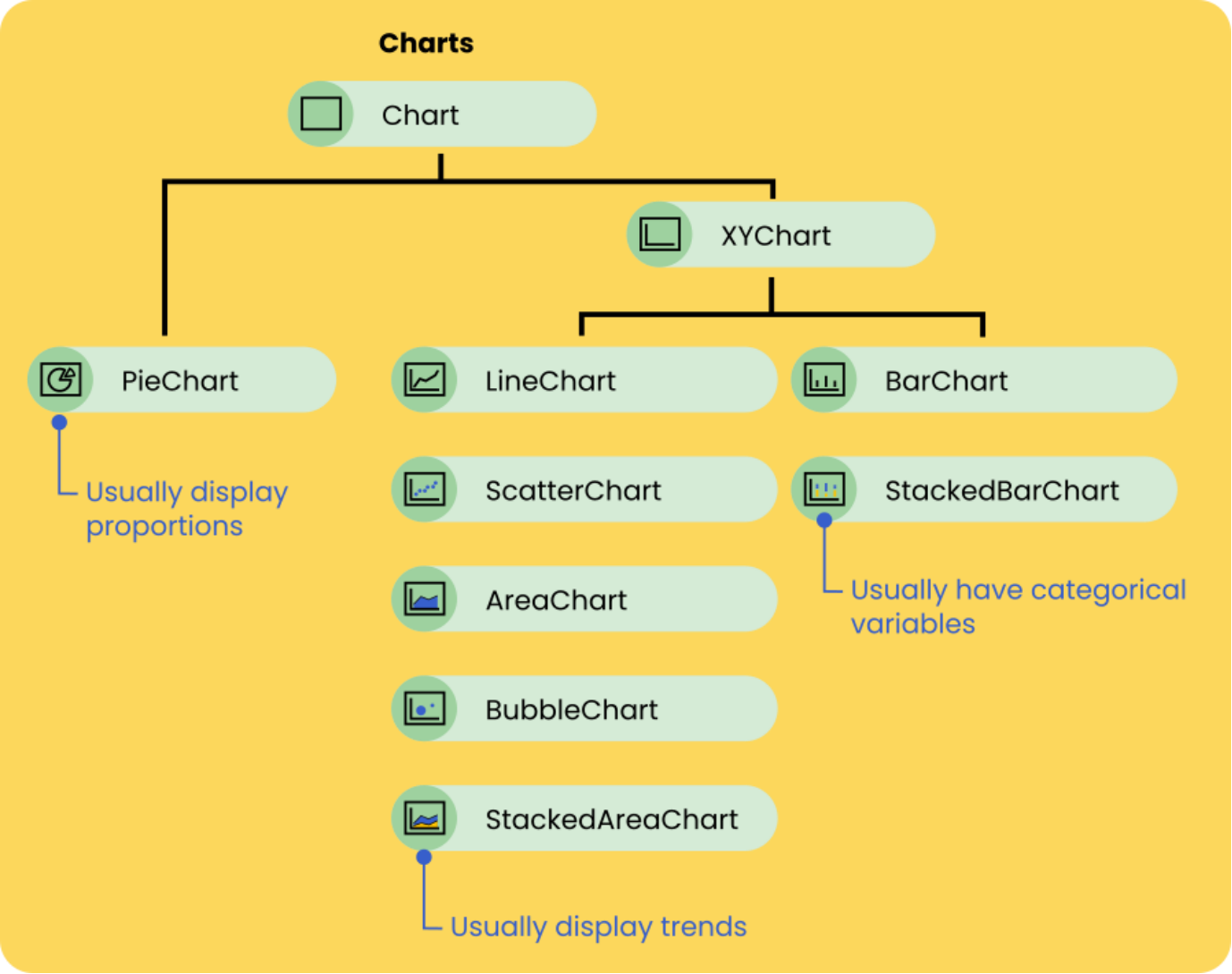
JavaFX Charts

- 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/



JavaFX Shape

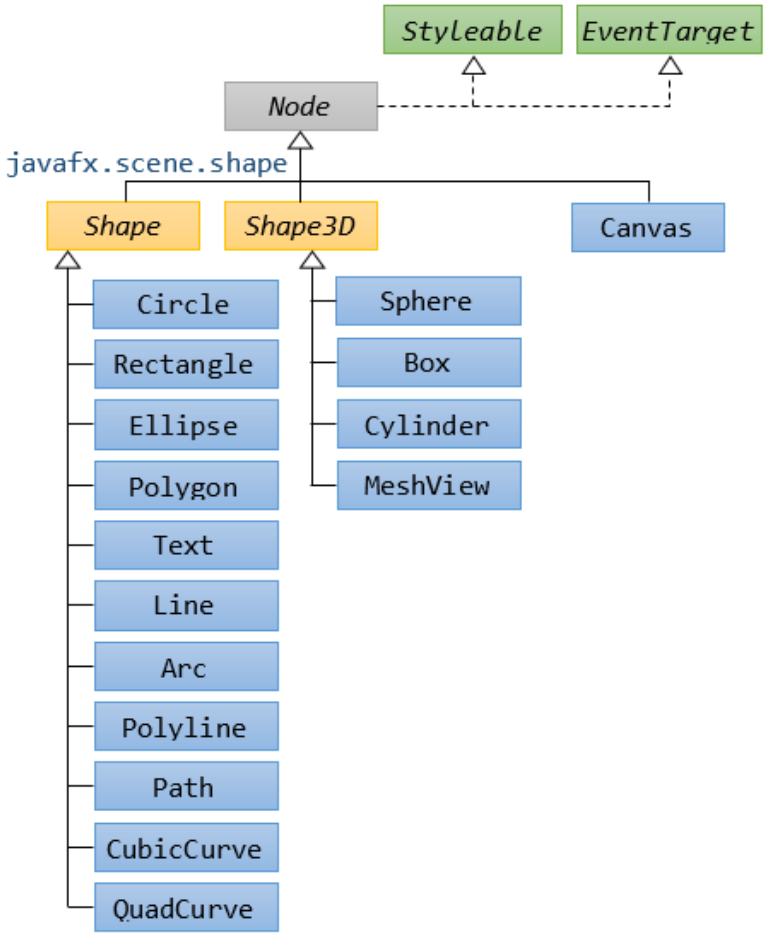
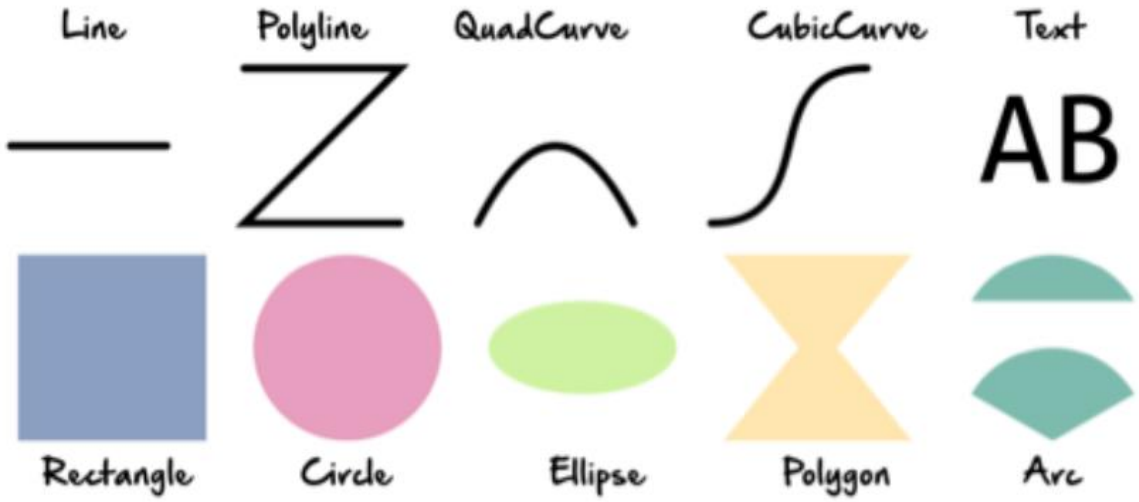
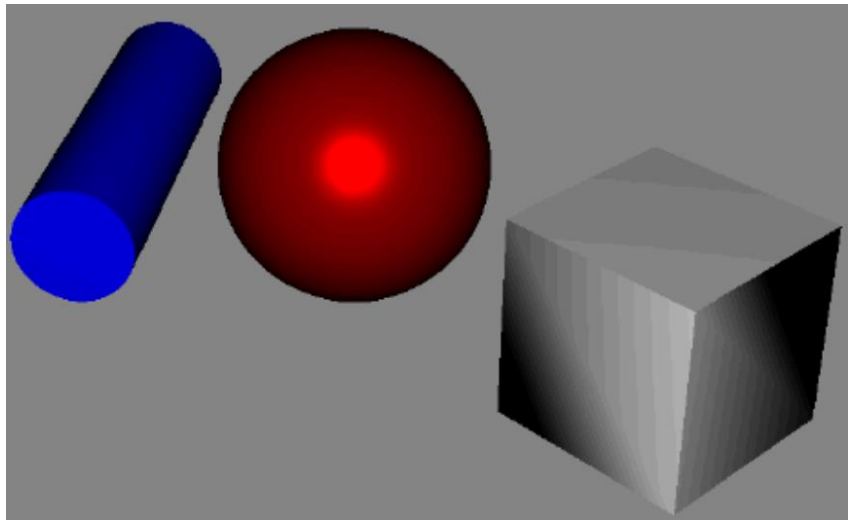


Image source: https://dzone.com/refcardz/javafx-8-1

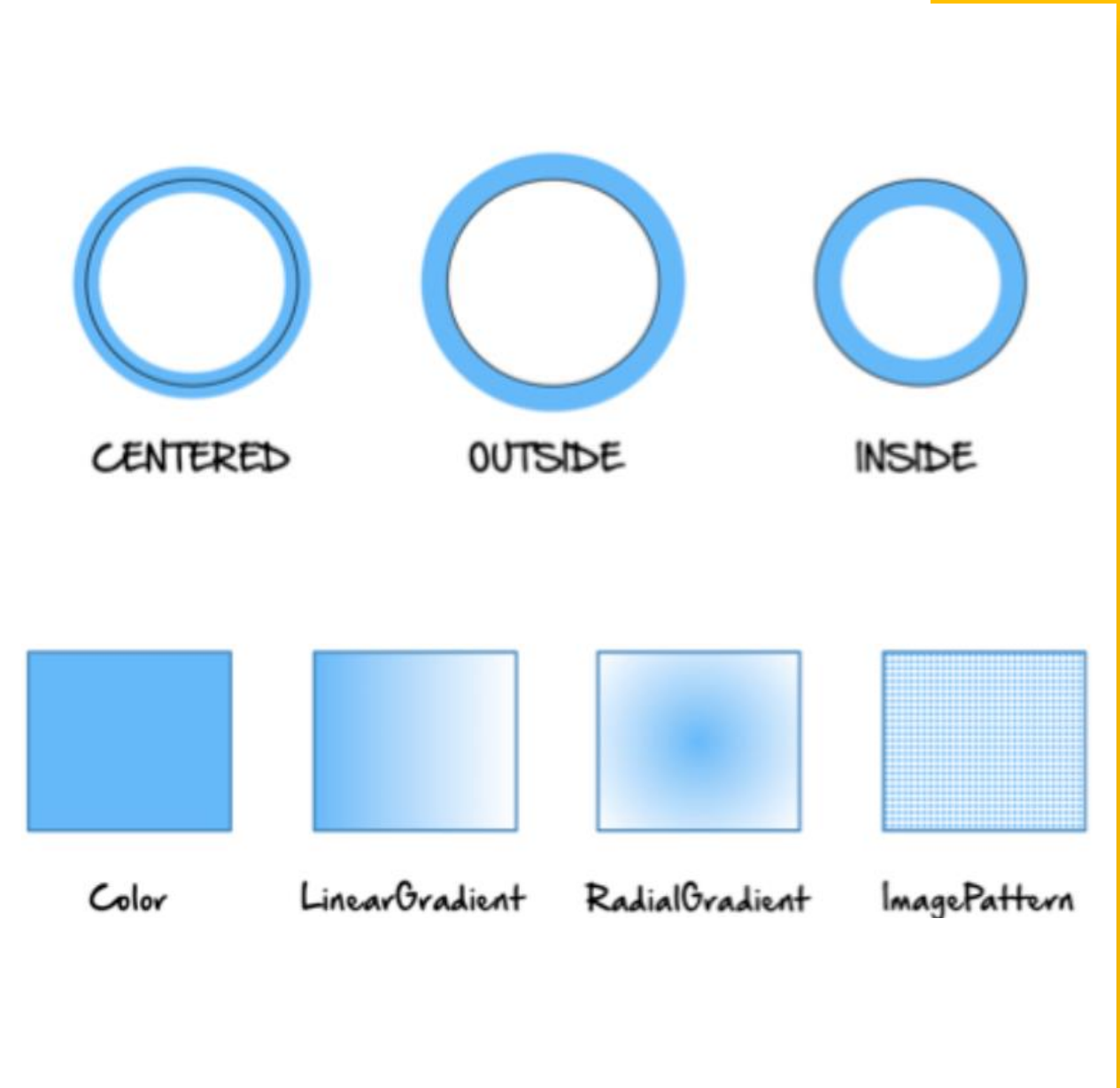




Shape Properties

- Fill
- Stroke/Outline
- Decoration styles

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



Shape Operations

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

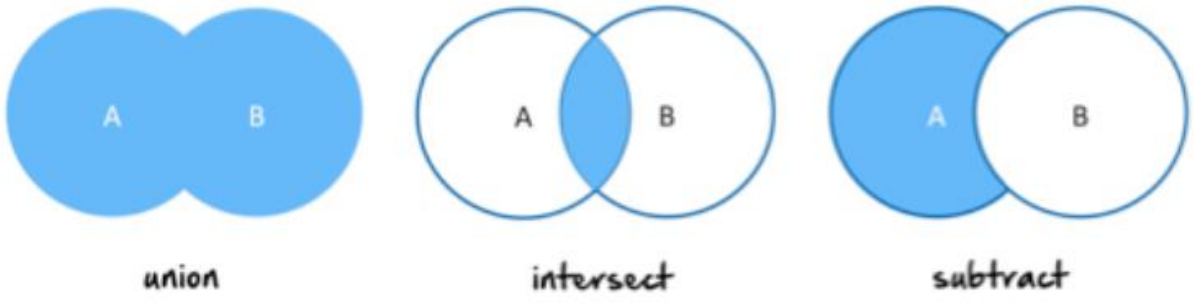
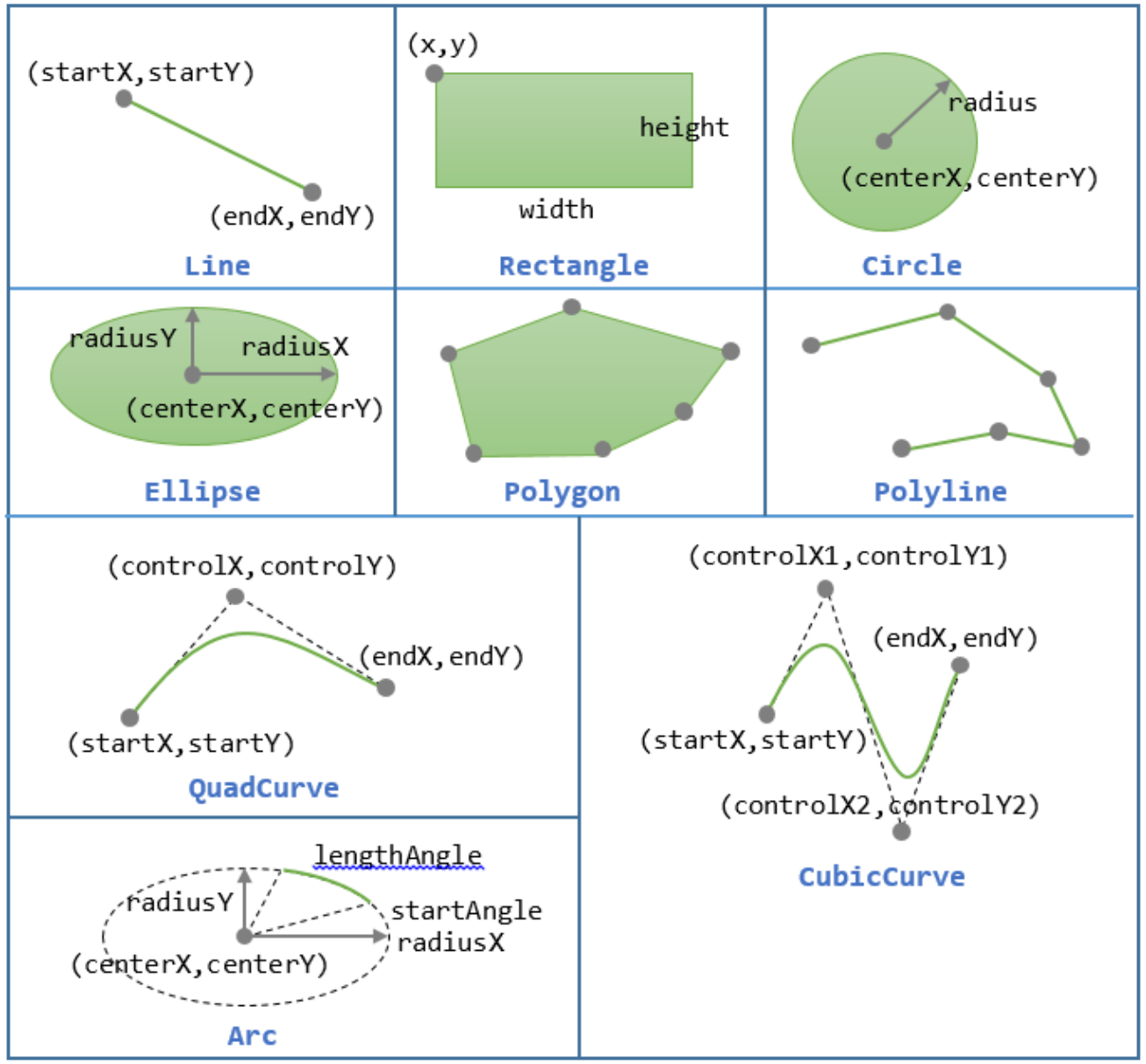


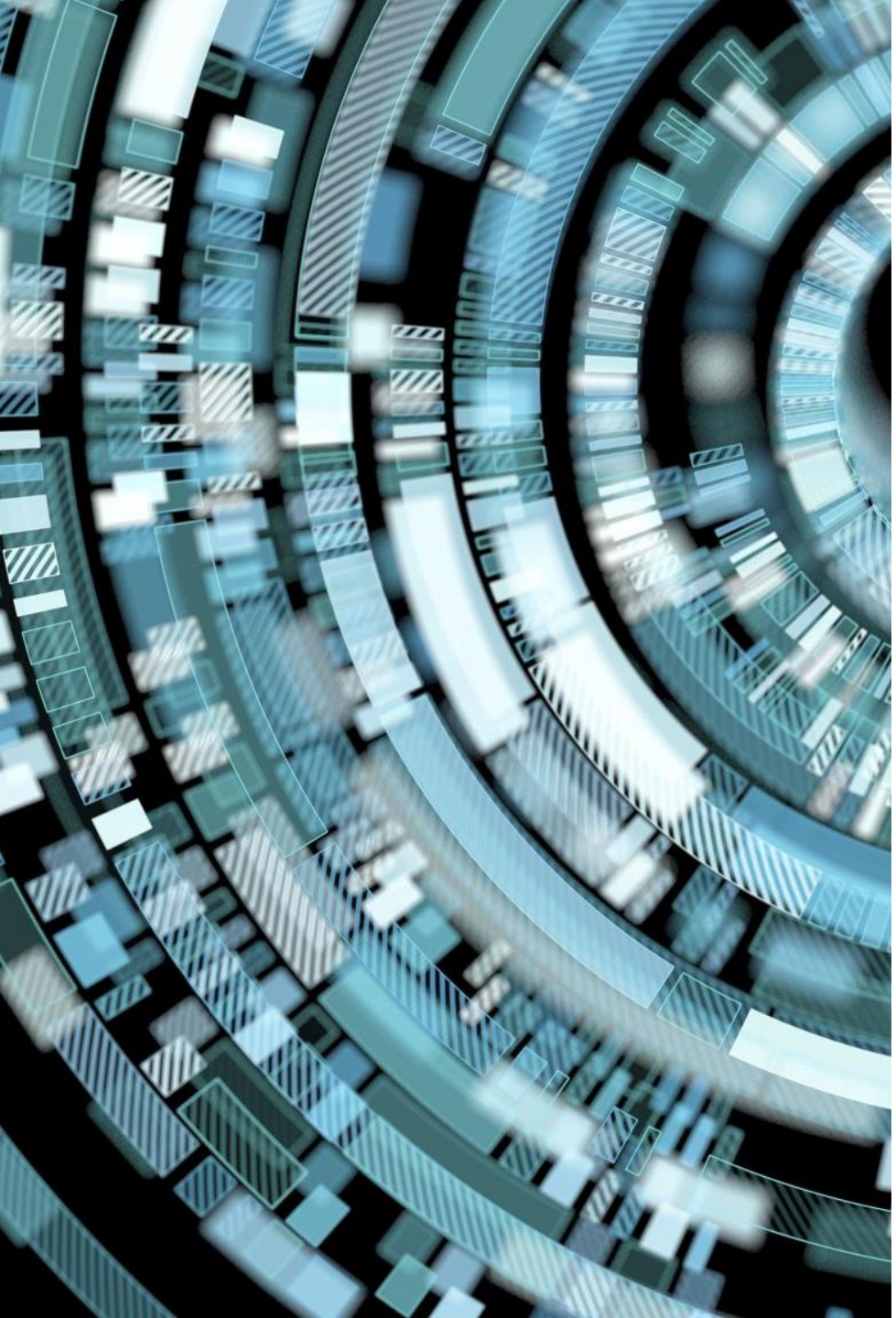
Image source: https://dzone.com/refcardz/javafx-8-1

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);
```

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





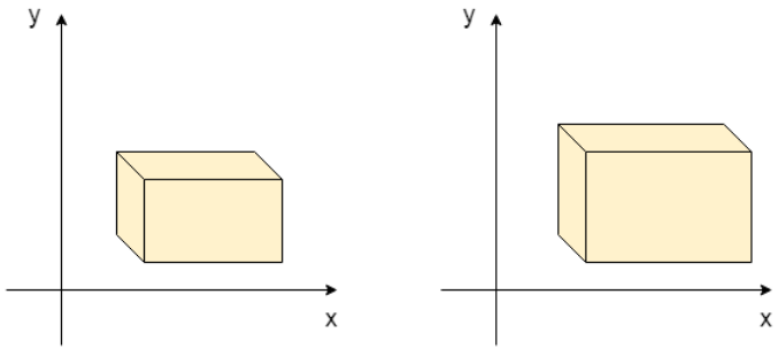
Lecture 9

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

JavaFX Transformation

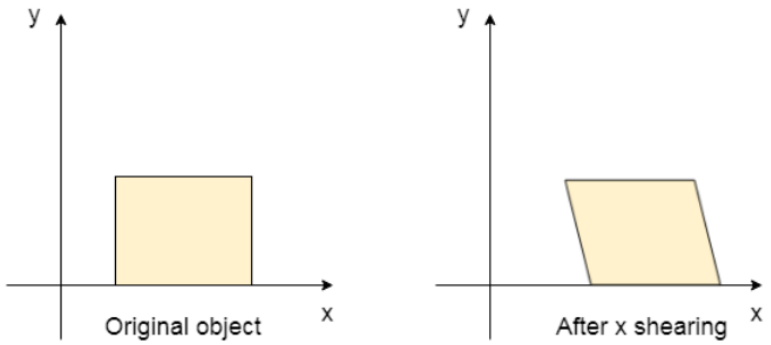
A transformation changes the place of a graphical object in a coordinate system according to certain parameters.

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



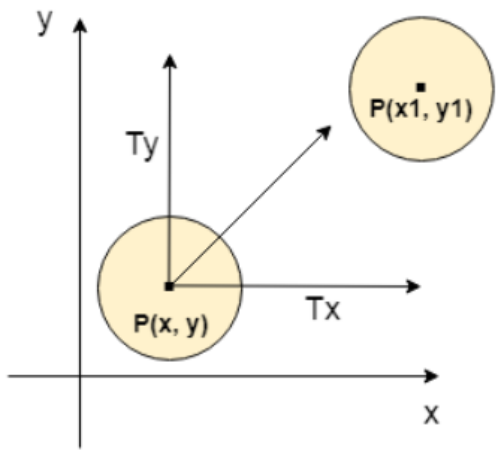
Scaling

Change the size of an object



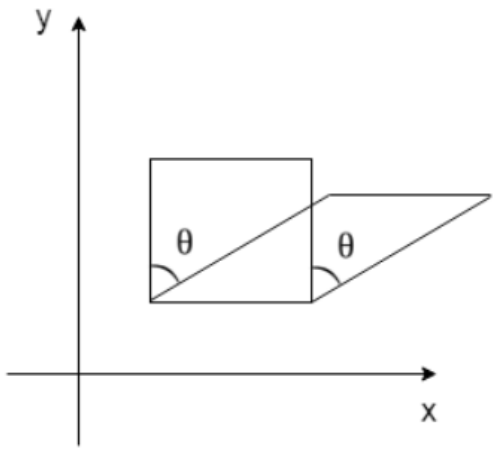
Shearing

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



Translation

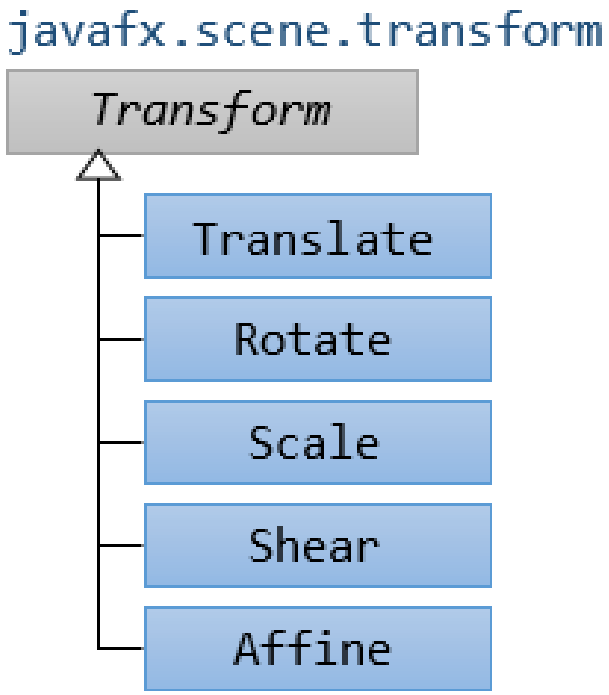
Change in the position of an object



Rotation

Rotate an object by a certain angle θ

JavaFX Transformation



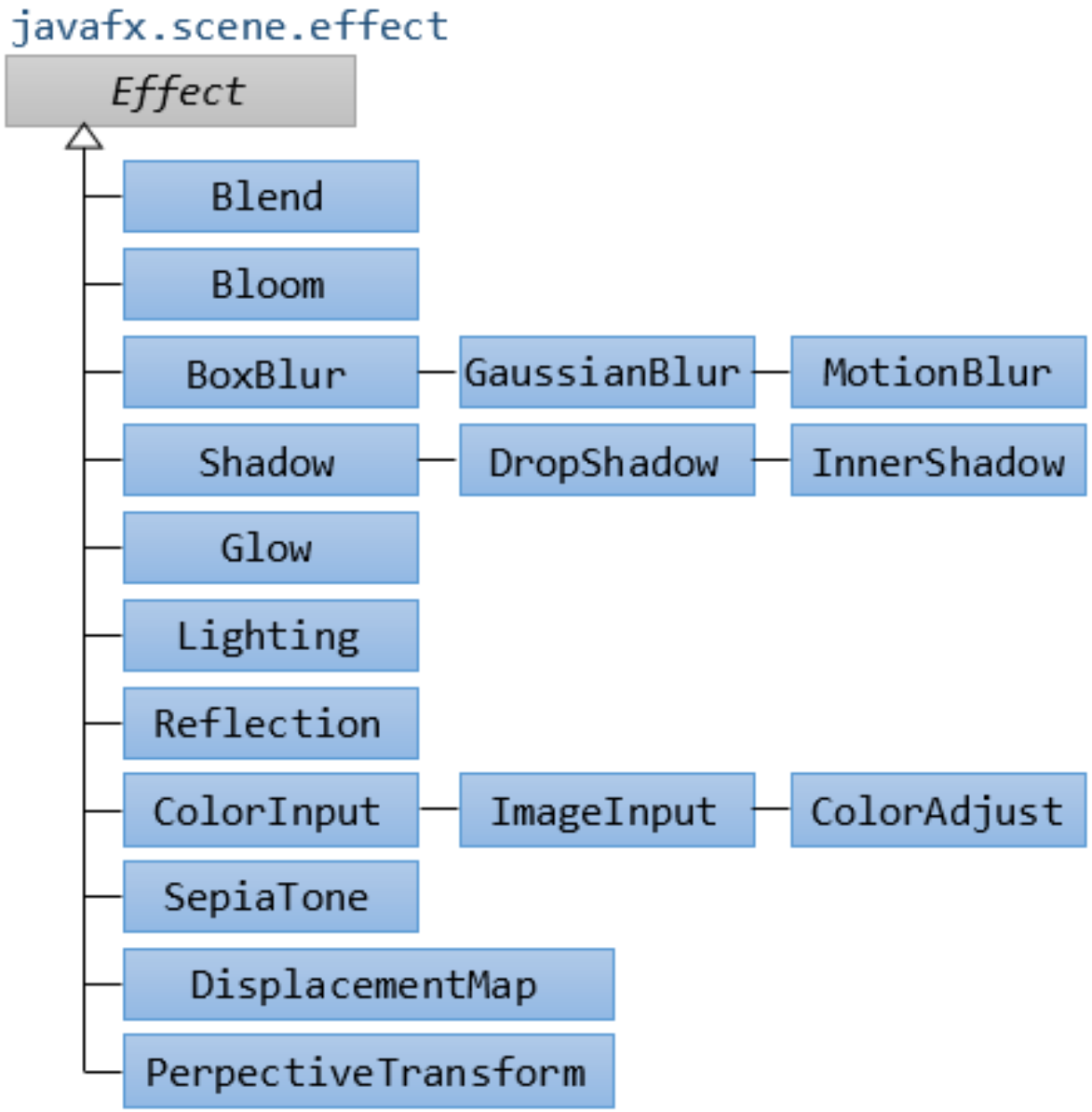
- All transformations are represented by various classes in package `javafx.scene.transform`
- `Transform` is the base class for different transformations

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

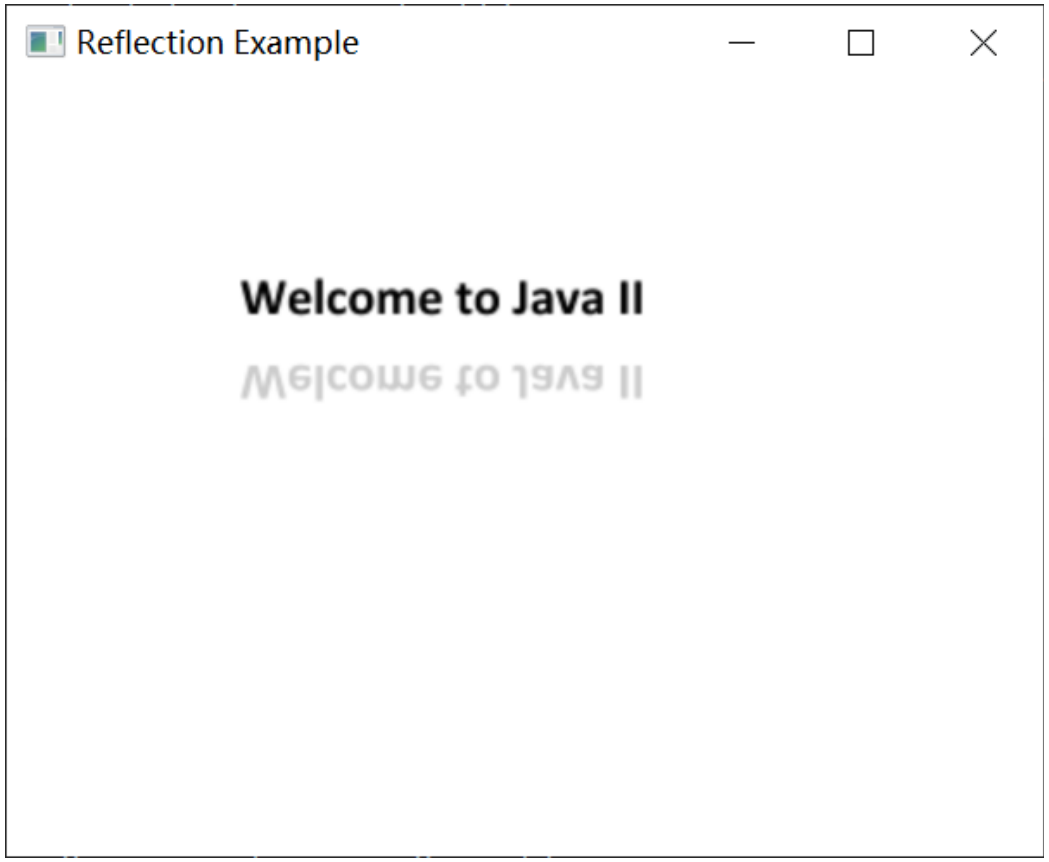
JavaFX Effects

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

JavaFX Effects



```java
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

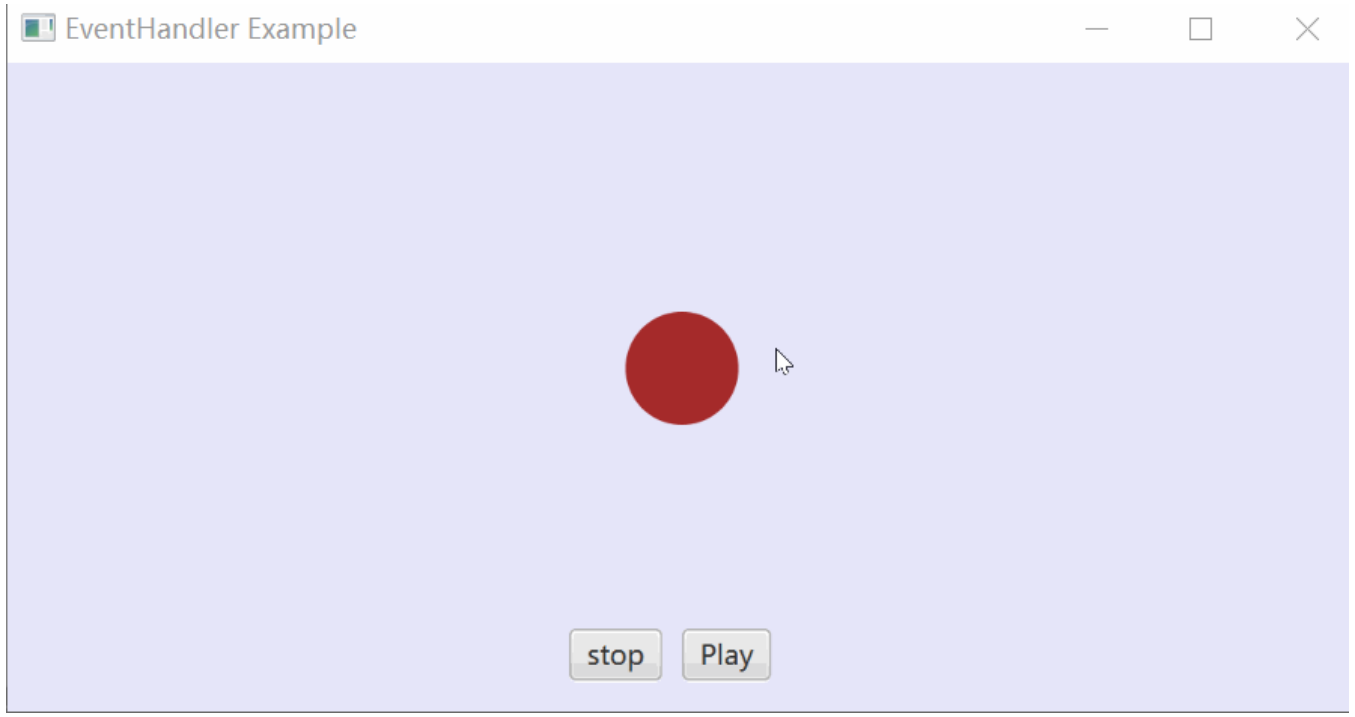
Class Transition

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();

//Moving to the staring point
MoveTo moveTo = new MoveTo( x: 208,  y: 71);
//Creating line path to a new point
LineTo line1 = new LineTo( x: 421,  y: 161);
LineTo line2 = new LineTo( x: 226, y: 232);
LineTo line3 = new LineTo( x: 332, y: 52);
LineTo line4 = new LineTo( x: 369,  y: 250);
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);
```

Extends Shape

PathElements
Drawing a straight line from the current coordinate to the new coordinates.

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

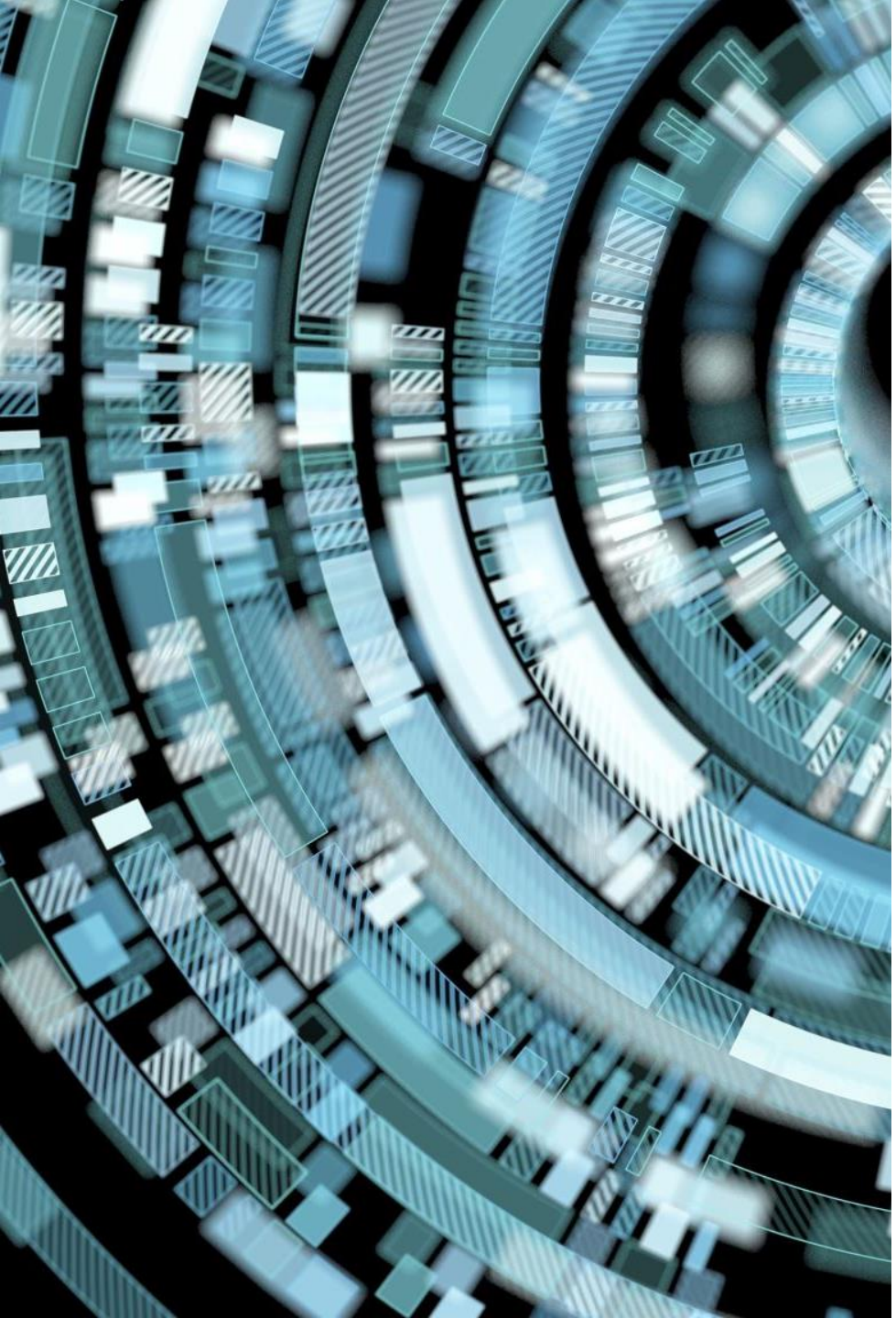
Add the Animation Event

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

play() is inherited from the Animation class

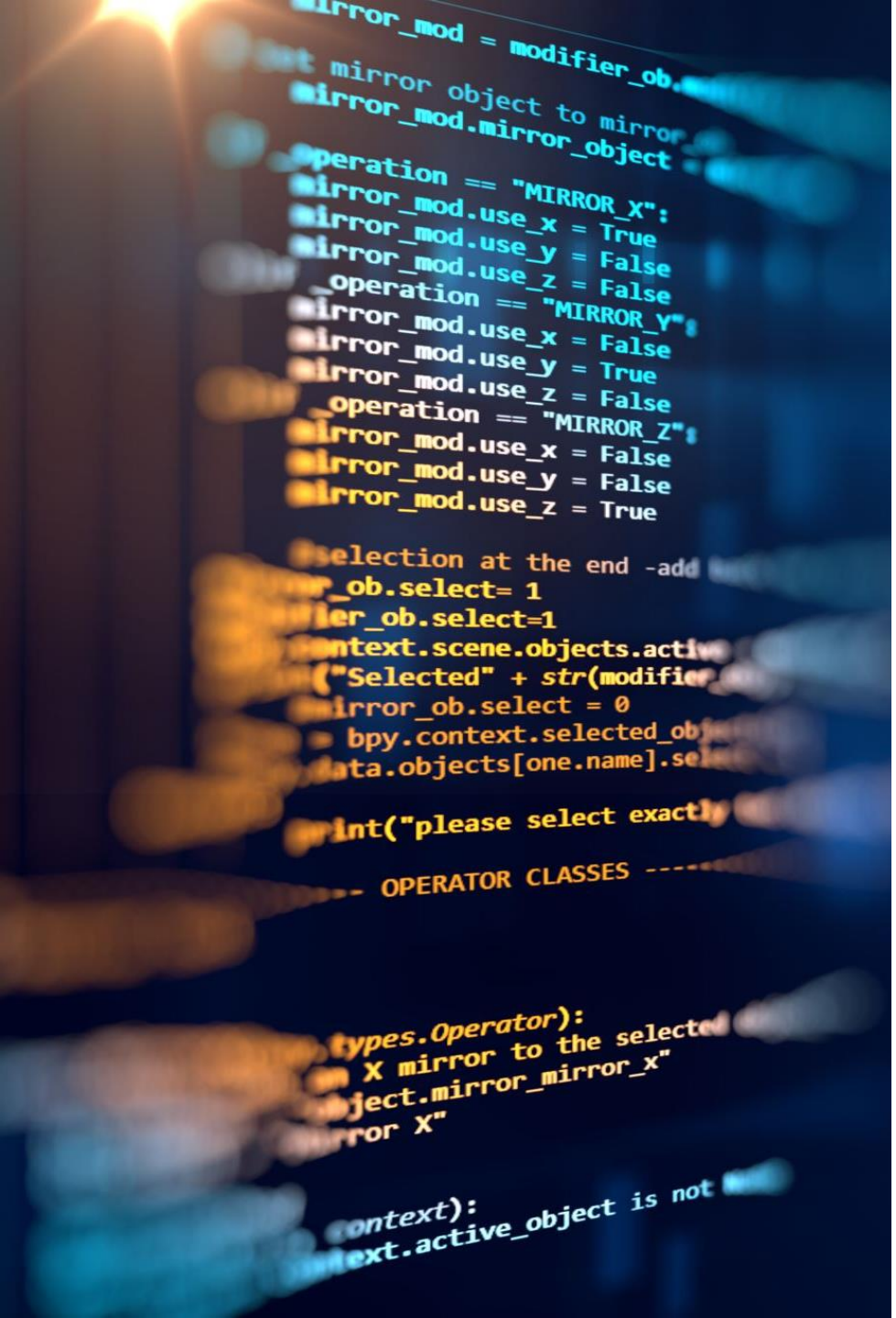
When button is clicked, play the animation

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

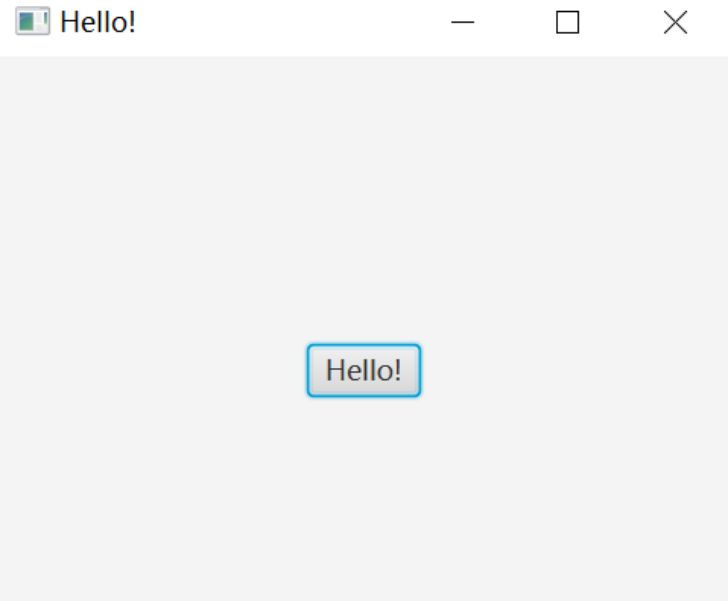
```xml
<?xml version="1.0" encoding="UTF-8"?>

<?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

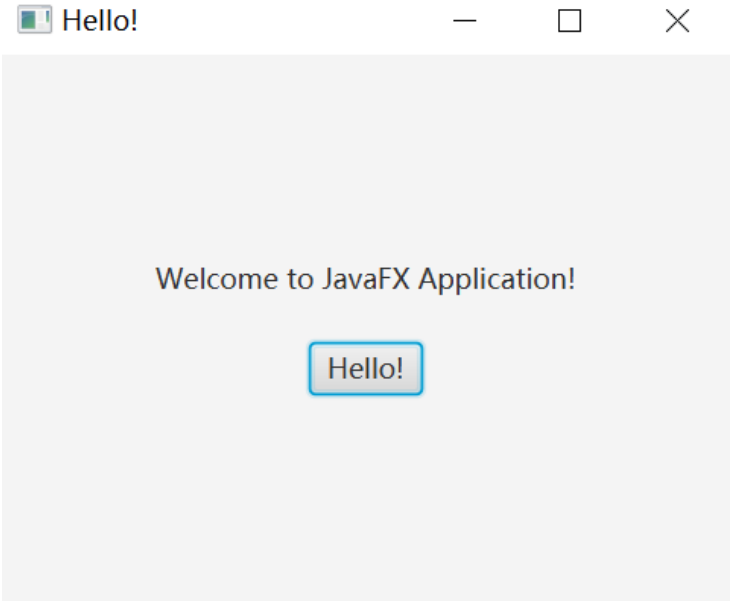
```java
import javafx.fxml.FXML;
import javafx.scene.control.Label;

public class HelloController {
    @FXML
    private Label welcomeText;

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

HelloController.java

Bootstrap JavaFX Application



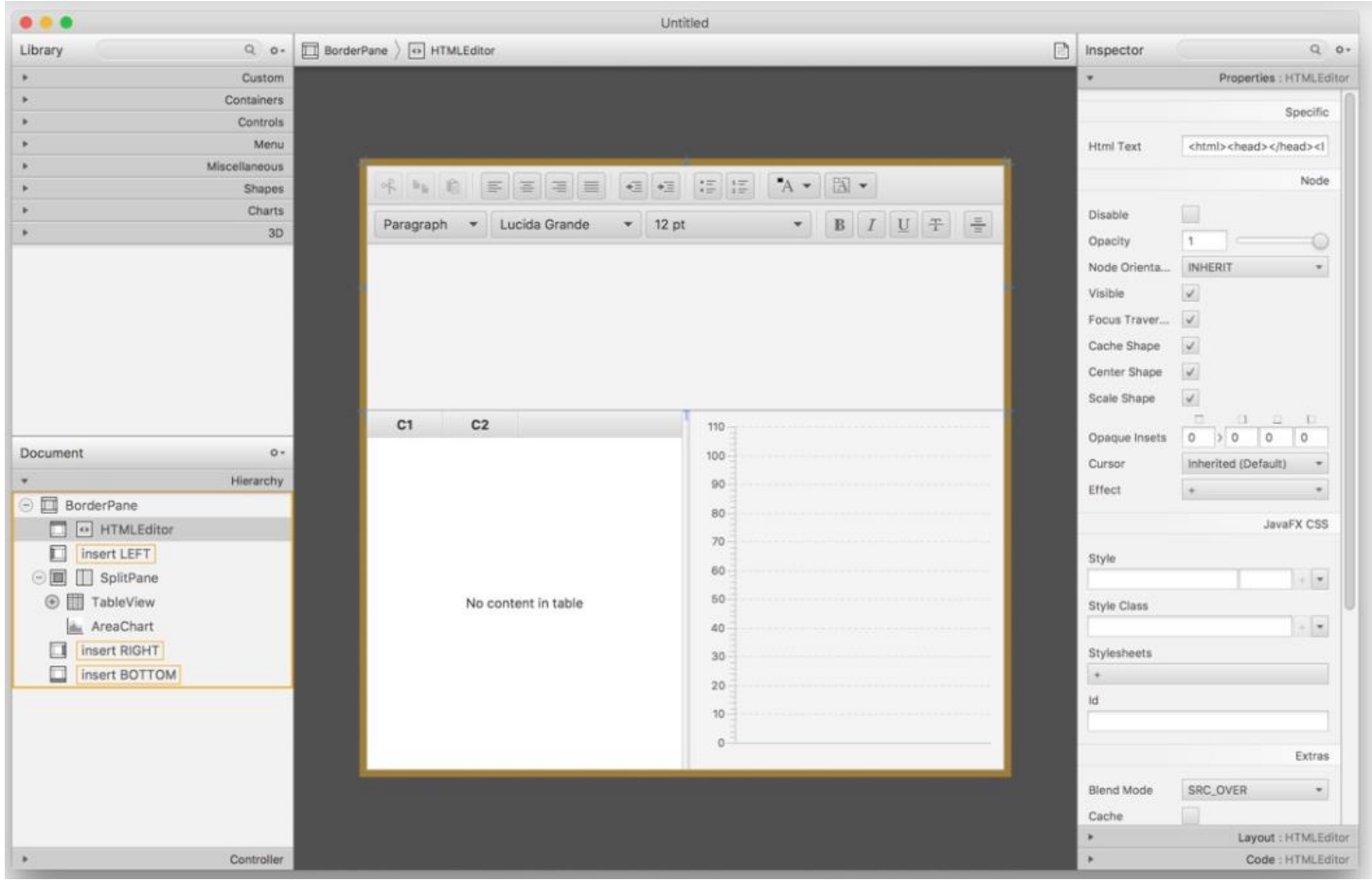
On start



Button clicked

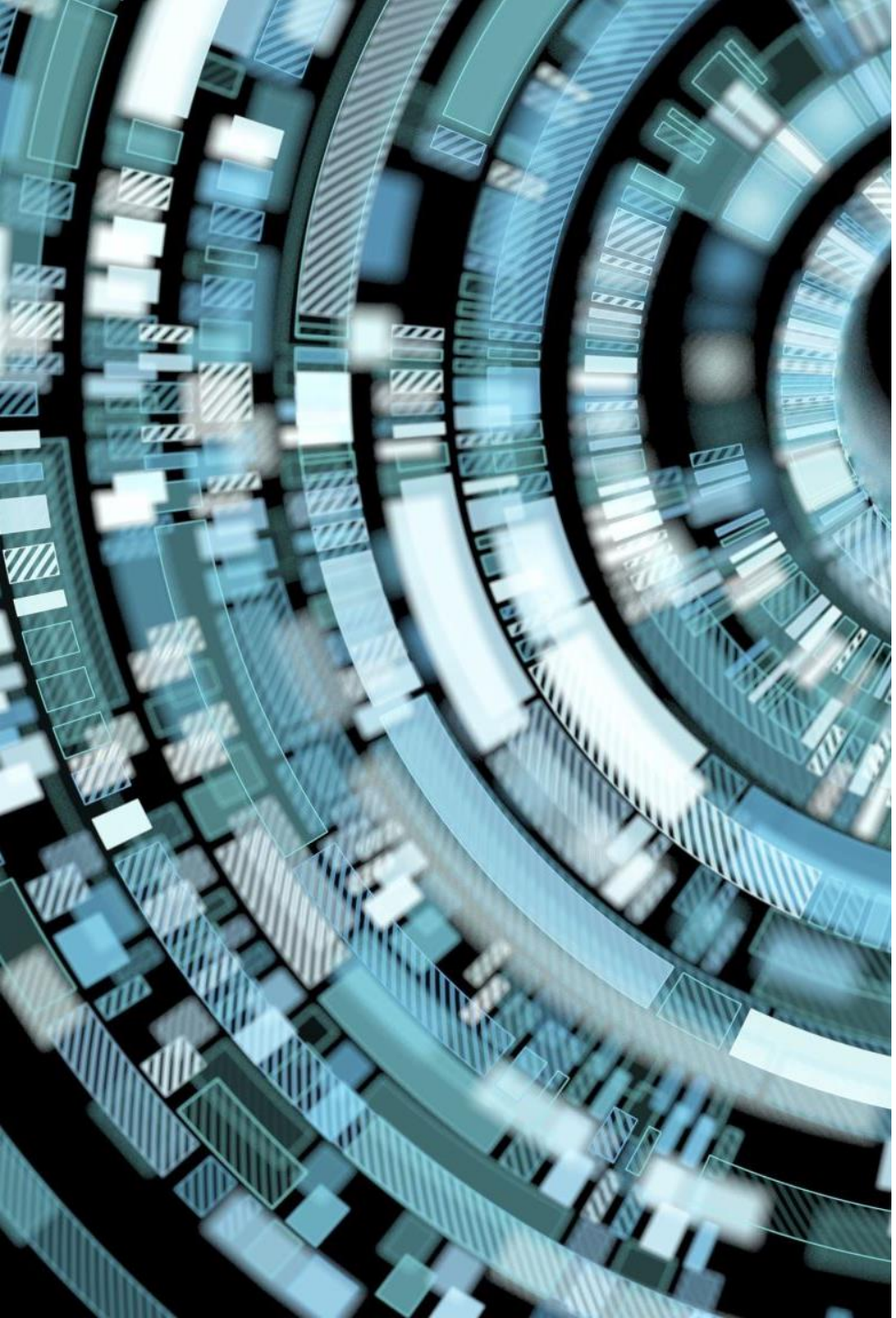
JavaFX Scene Builder

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



Drag & Drop,
Rapid Application
Development.

Download Now



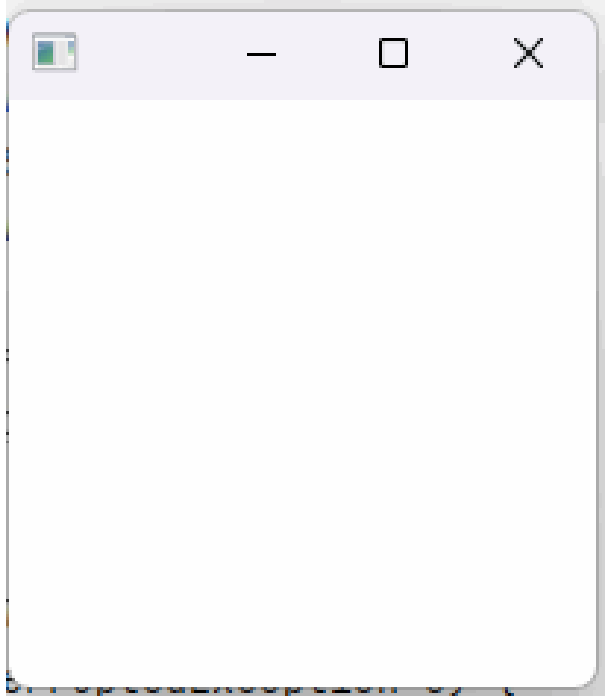
Lecture 9

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

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.

JavaFX Threading



```java
public class Concurrency extends Application {
    @Override
    public void start(Stage primaryStage) {

        ProgressBar progressBar = new ProgressBar(0);
        VBox vBox = new VBox(progressBar);
        Scene scene = new Scene(vBox, 200, 200);
        primaryStage.setScene(scene);
        primaryStage.show();

        double progress = 0;
        for (int i = 0; i < 10; i++) {
            try {
                // mimic long-running task
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            progress += 0.1;
            double reportedProgress = progress;
            progressBar.setProgress(reportedProgress);
        }
    }
}
```

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

JavaFX Threading

- 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.

```java
public class Concurrency extends Application {
    @Override
    public void start(Stage primaryStage) {

        ProgressBar progressBar = new ProgressBar(0);
        VBox vBox = new VBox(progressBar);
        Scene scene = new Scene(vBox, 200, 200);
        primaryStage.setScene(scene);
        primaryStage.show();

        double progress = 0;
        for (int i = 0; i < 10; i++) {
            try {
                // mimic long-running task
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            progress += 0.1;
            final double reportedProgress = progress;
            progressBar.setProgress(reportedProgress);
        }
    }
}
```

JavaFX Threading

- Generally, tasks should not interact directly with the UI.
- Doing so creates a **tight coupling** between a specific Task implementation and a specific part of your UI.

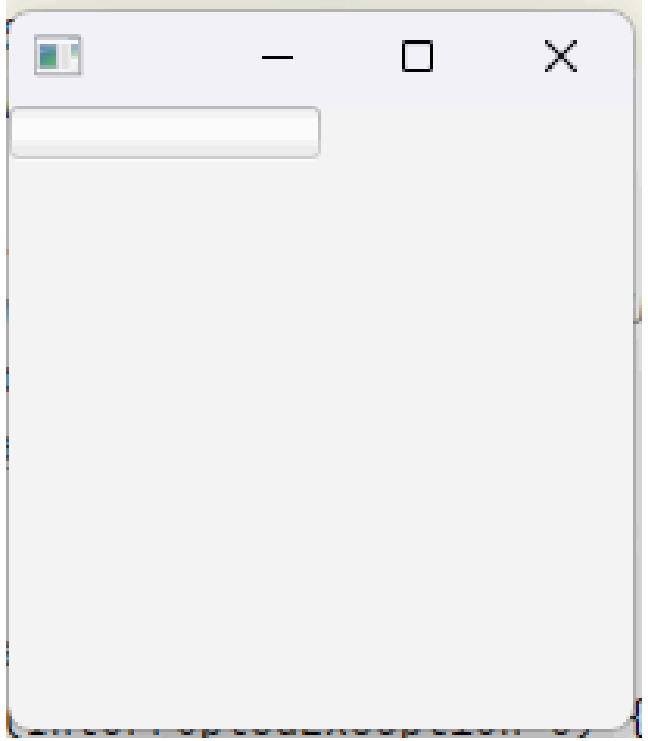
```java
public class Concurrency extends Application {
    @Override
    public void start(Stage primaryStage) {

        ProgressBar progressBar = new ProgressBar(0);
        VBox vBox = new VBox(progressBar);
        Scene scene = new Scene(vBox, 200, 200);
        primaryStage.setScene(scene);
        primaryStage.show();

        double progress = 0;
        for (int i = 0; i < 10; i++) {
            try {
                // mimic long-running task
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            progress += 0.1;
            final double reportedProgress = progress;
            progressBar.setProgress(reportedProgress);
        }
    }
}
```

JavaFX Threading

- We could also run long-running tasks on one or more background threads
- The background thread informs the JavaFX Application thread by `Platform.runLater()` 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.

```java
public class Concurrency extends Application {
    @Override
    public void start(Stage primaryStage) {

        ProgressBar progressBar = new ProgressBar(0);
        VBox vBox = new VBox(progressBar);
        Scene scene = new Scene(vBox, 200, 200);
        primaryStage.setScene(scene);
        primaryStage.show();

        Thread taskThread = new Thread(() -> {
            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(() -> progressBar
                        .setProgress(reportedProgress));
            }
        });

        taskThread.start();
    }
}
```

JavaFX Threading



```java
public class Concurrency extends Application {
    @Override
    public void start(Stage primaryStage) {

        ProgressBar progressBar = new ProgressBar(0);
        VBox vBox = new VBox(progressBar);
        Scene scene = new Scene(vBox, 200, 200);
        primaryStage.setScene(scene);
        primaryStage.show();

        Thread taskThread = new Thread(() -> {
            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(() -> progressBar
                        .setProgress(reportedProgress));
            }
        });

        taskThread.start();
    }
}
```

JavaFX Threading

- The `javafx.concurrent` package leverages the existing `java.util.concurrent` package
- The `Task` class enables developers to implement asynchronous tasks in JavaFX applications.
- The `Service` class executes tasks.

A task can be started in one of the following ways:

- By starting a thread with the given task as a parameter:

```
Thread th = new Thread(task);
th.setDaemon(true);
th.start();
```

- By using the `ExecutorService` API:

```
ExecutorService.submit(task);
```

https://docs.oracle.com/javafx/2/threads/jfxpub-threads.htm

Next Lecture

- Reflections
- Annotations