FPT Education

FPT UNIVERSITY

Java

Introduction to JavaFX

Objectives

- Understand JavaFX Architecture
- JavaFX Core: Stage, Scene, FXML
- Understand JavaFX Properties and Bindings, Events, Layouts, Controls
- Create Java desktop applications with JavaFX

What is JavaFX?

- JavaFX is an open source, next generation client application platform for desktop, mobile and embedded systems built on Java.
- It is a collaborative effort by many individuals and companies with the goal of producing a modern, efficient, and fully featured toolkit for developing rich client applications.
- A powerful framework for building rich client applications in Java.
- Successor to Swing, offering a modern approach to GUI development.
- Open-source and available as a separate library from Java 11 onwards.

JavaFX - Key features

- Written in Java: Leverages the power and flexibility of the Java language, including multithreading, lambda expressions, etc.
- Declarative UI with FXML: Define your UI structure using XML, making it easy to separate UI design from application logic.
- Data Binding: Simplifies data synchronization between UI elements and the underlying data model.
- Rich UI Controls: Provides a diverse set of modern and customizable UI controls, including buttons, text fields, charts, and more.

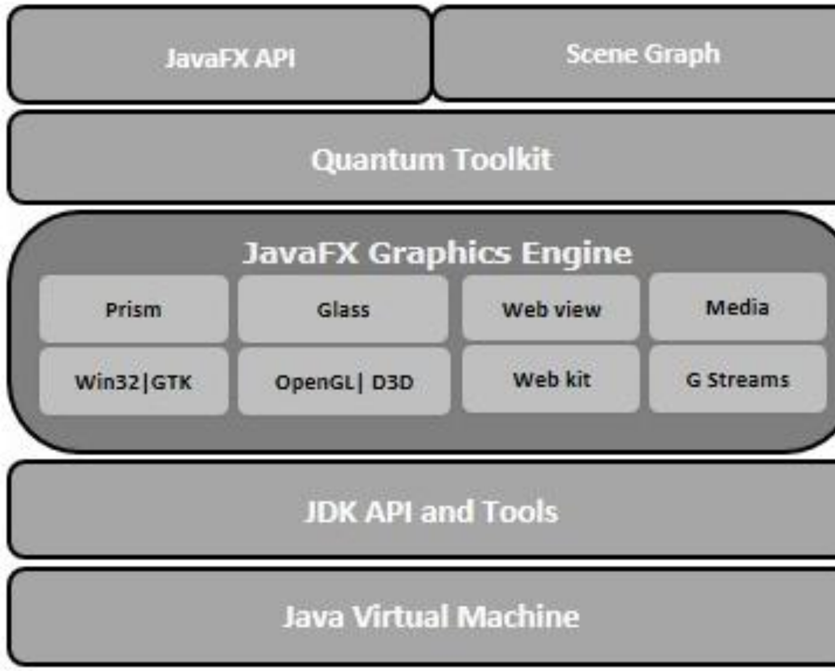
JavaFX - Key features

- Media and Web Integration: Enables playback of audio and video content and embedding web components within your application.
- 2D and 3D Graphics: Offers robust support for both 2D and 3D graphics rendering, allowing you to create visually stunning applications.
- Animations and Effects: Create smooth animations and apply various visual effects to enhance user experience.
- CSS Styling: Style your application using CSS for consistent and flexible design.

FPT Education
FPT UNIVERSITY

JavaFX - Architecture

- **javafx.animation** − Contains classes to add transition based animations such as fill, fade, rotate, scale and translation, to the JavaFX nodes.
- **javafx.application** − Contains a set of classes responsible for the JavaFX application life cycle.
- **javafx.css** − Contains classes to add CSS–like styling to JavaFX GUI applications.



JavaFX - Architecture

- **javafx.event** − Contains classes and interfaces to deliver and handle JavaFX events.
- **javafx.geometry** − Contains classes to define 2D objects and perform operations on them.
- **javafx.stage** − This package holds the top level container classes for JavaFX application.
- **javafx.scene** − This package provides classes and interfaces to support the scene graph. In addition, it also provides sub-packages such as canvas, chart, control, effect, image, input, layout, media, paint, shape, text, transform, web, etc.

JavaFX Parts

- JavaFX has 3 parts
  - A GUI builder called **SceneBuilder** allows drag-and-drop manipulation of widgets.
  - A configuration language called **FXML** that records the widgets in the GUI, their visible attributes and their relationship to each other.
  - A **Controller** class that must be completed by the programmer to bring the GUI to life.
- A JavaFX application has some additional parts
  - A set of classes to describe the model, which is what the GUI allows the user to interact with.
  - A set of cascading style sheets (**CSS** files) to further specify “look-and-feel”.

JavaFX Scene Builder

- JavaFX Scene Builder is a standalone JavaFX GUI visual layout tool that can also be used with various IDEs including Eclipse, NetBeans and IntelliJ.
- JavaFX Scene Builder enables you to create GUIs by dragging and dropping GUI components from Scene Builder's library onto a design area, then modifying and styling the GUI—all without writing any code.
- JavaFX Scene Builder generates FXML (FX Markup Language)—an XML vocabulary for defining and arranging JavaFX GUI controls without writing any Java code.

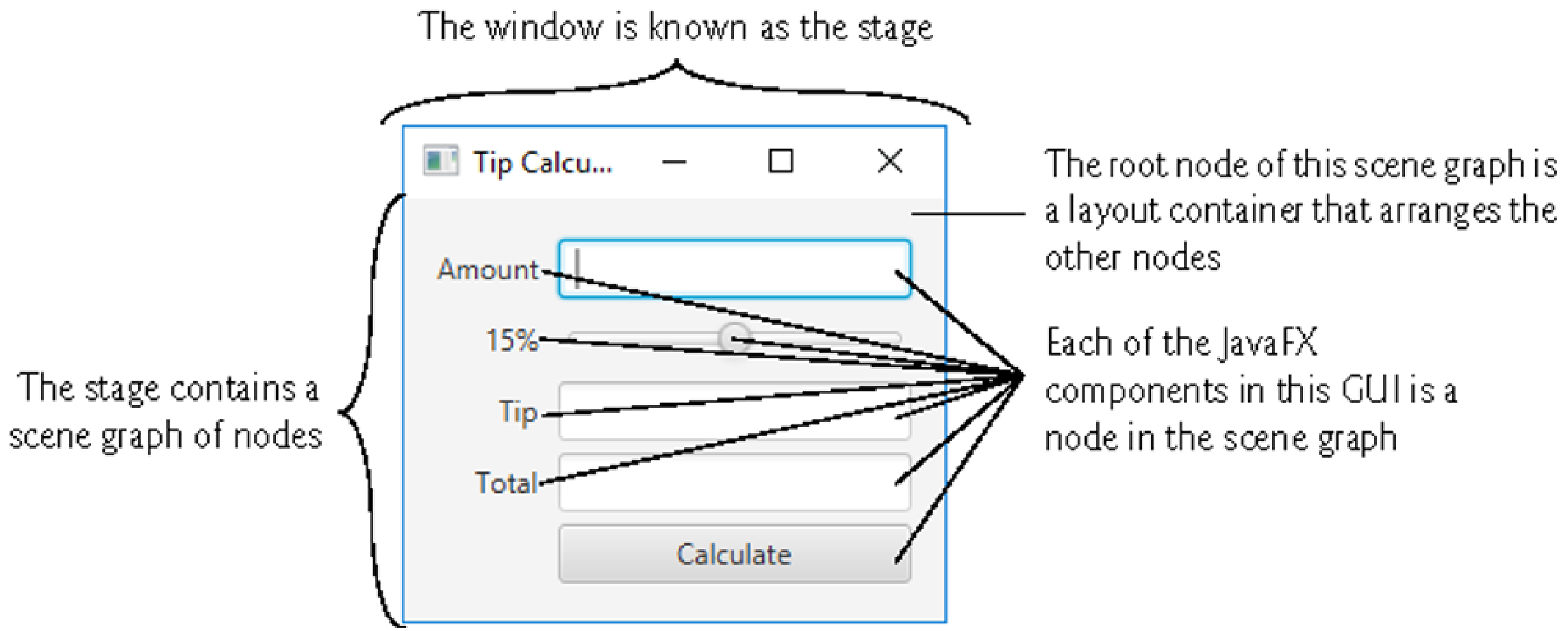
JavaFX Scene Builder

- The FXML code is separate from the program logic that’s defined in Java source code—this separation of the interface (the GUI) from the implementation (the Java code) makes it easier to debug, modify and maintain JavaFX GUI apps.

- Placing GUI components in a window can be tedious. Being able to do it dynamically using a configuration file makes the job much easier.

- No additional compilation is needed unless actions need to be programmed in the Controller.java class.

JavaFX App Window Structure


The window is known as the stage
Tip Calcu...
Amount
15%
Tip
Total
Calculate
The root node of this scene graph is a layout container that arranges the other nodes
Each of the JavaFX components in this GUI is a node in the scene graph
The stage contains a scene graph of nodes

JavaFX App Window Structure

- The **Stage** is the window in which a JavaFX app's GUI is displayed
  - It's an instance of class **Stage** (package javafx.stage).
- The **Stage** contains one active **Scene** that defines the GUI as a **scene graph - a** tree data structure of an app's visual elements, such as GUI controls, shapes, images, video, text and.
- The scene is an instance of class **Scene** (package javafx.scene).

- **Controls** are GUI components, such as
  - Labels that display text,
  - TextFields that enable a program to receive user input,
  - Buttons that users click to initiate actions, and more.

JavaFX Application Layout

- An application Window in JavaFX is known as a **Stage**.
  - package javafx.stage

- A **Stage** contains an active **Scene** which is set to a Layout container.
  - package javafx.scene

- The Scene may have other Layout containers for organizing **Controllers** in a Tree organization.
  - Nodes with children are layout containers.
  - Nodes without children are widgets.

JavaFX Application Layout

- Each visual element in the scene graph is a **node** - an instance of a subclass of **Node** (package javafx.scene), which defines common attributes and behaviors for all nodes

- With the exception of the first node in the scene graph - the **root node** - each node in the scene graph has one parent.

- Nodes can have transforms (e.g., moving, rotating and scaling), opacity (whether a node is transparent, partially transparent or opaque), effects (e.g., drop shadows, blurs, reflection and lighting) and more.

JavaFX Application Controls

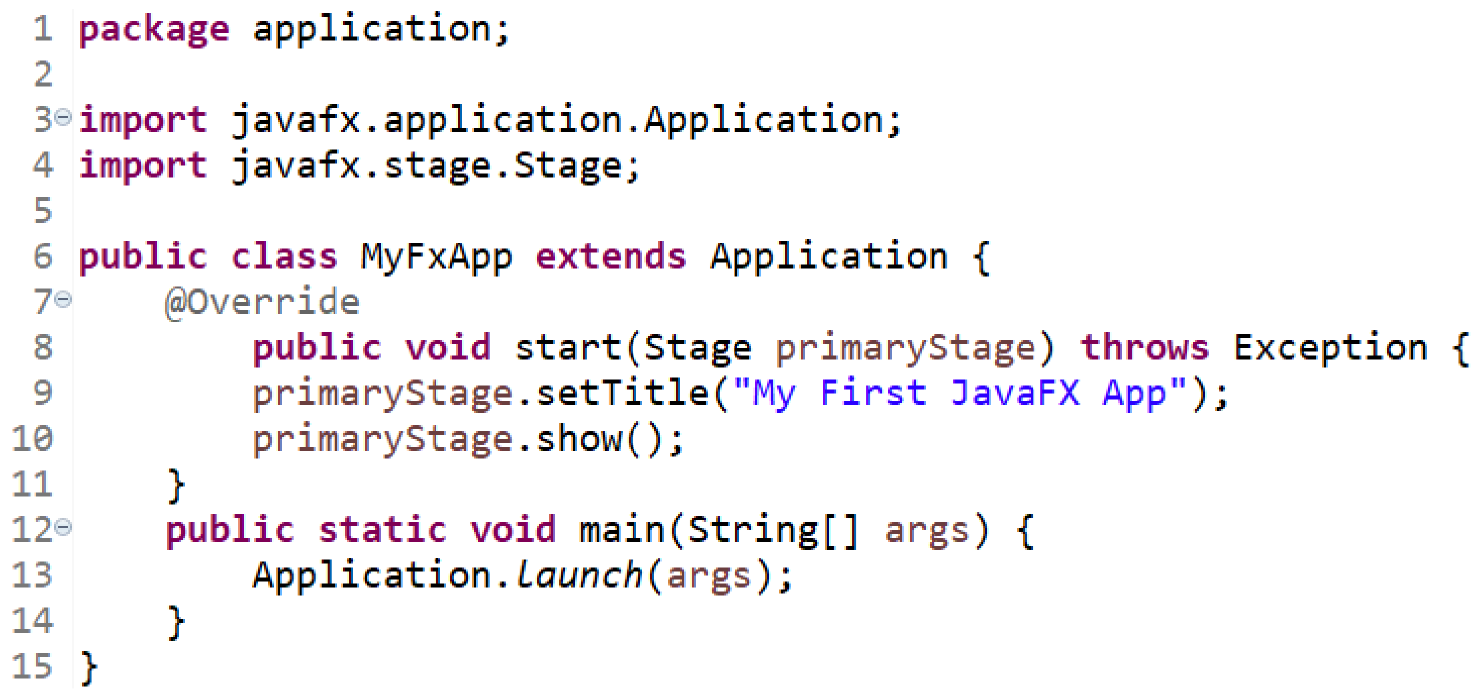
- Nodes with children are typically **layout containers** that arrange their child nodes in the scene.
  - **Layout containers** contain **controls** that accept inputs or other layout containers.

- When the user interacts with a **control**, the control generates an **event**.
- Programs can respond to these events—known as event handling—to specify what should happen when each user interaction occurs.

- An **event handler** is a method that responds to a user interaction. An FXML GUI's event handlers are defined in a so-called **controller class**.

JavaFX Application Class

- JavaFX class must implement the abstract **start()** method of the **Application** class

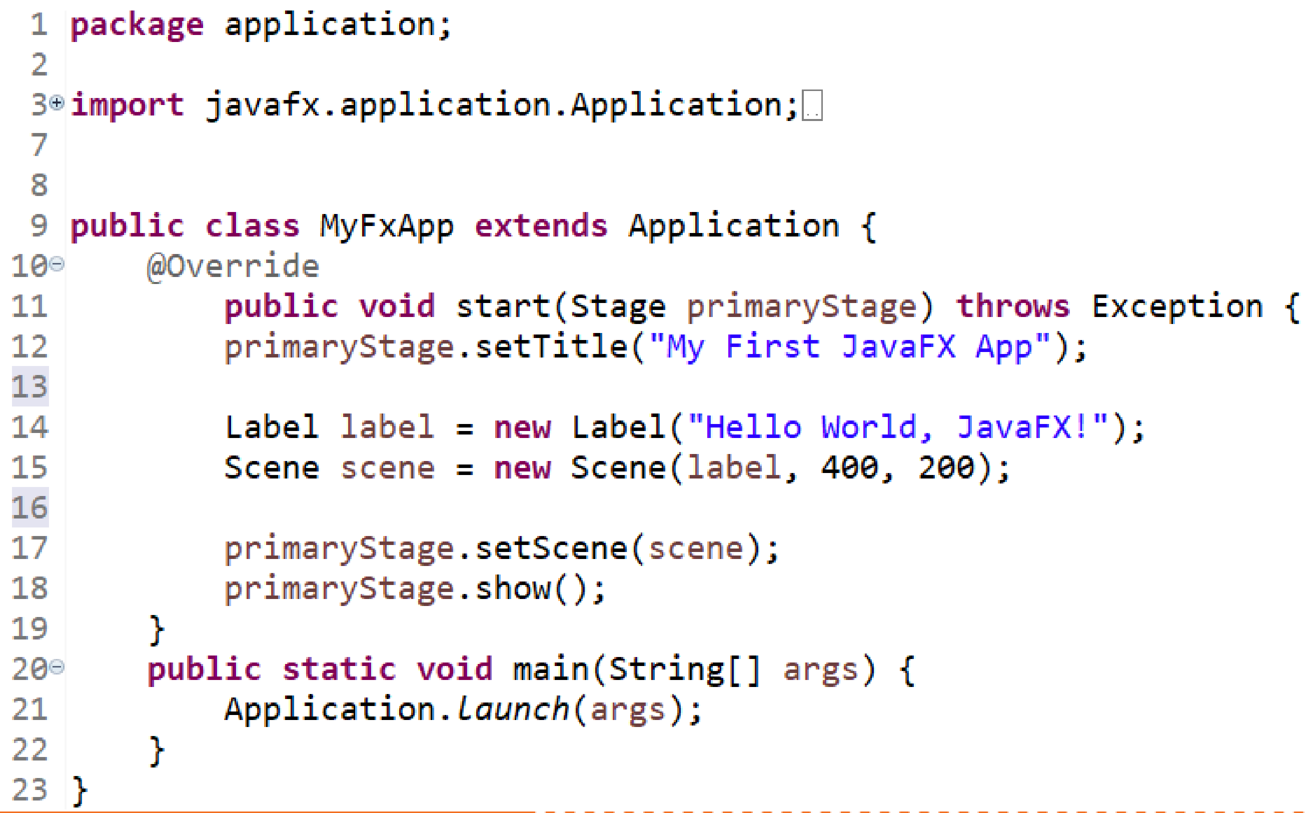


```java
package application;

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);
    }
}
```

JavaFX Core Components, Concepts and Features

Scene to the Stage object

- To display something inside the JavaFX application window you must add a Scene to the Stage object.



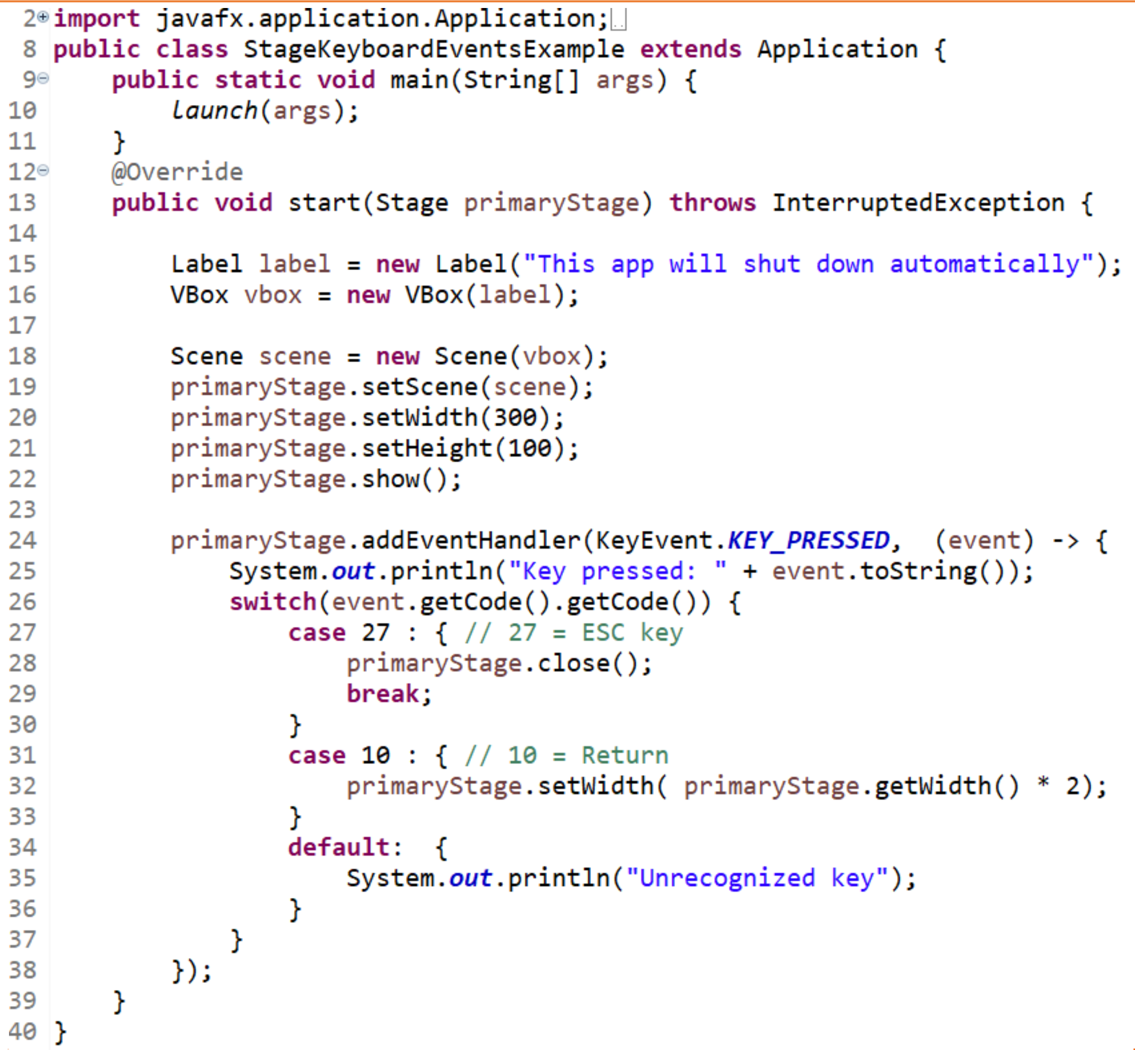
```java
package application;

import javafx.application.Application;


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

        Label label = new Label("Hello World, JavaFX!");
        Scene scene = new Scene(label, 400, 200);

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

JavaFX Stage

- A JavaFX Stage, javafx.stage.Stage, represents a window in a JavaFX desktop application. Inside a JavaFX Stage, insert a JavaFX Scene which represents the content displayed inside a window - inside a Stage.
- Operations:
  - Creating a Stage
  - Showing a Stage (show() vs. showAndWait())
  - Set a Scene on a Stage
  - Stage Title
  - Stage Position
  - Stage Width and Height
  - Stage Modality
  - Stage Owner
  - Stage Style (DECORATED, UNDECORATED, TRANSPARENT, UNIFIED, UTILITY)

Stage Life Cycle Events

- The Stage events: Close Request, Hiding, Hidden, Showing, Shown
- Stage keyboard events



```java
import javafx.application.Application;
public class StageKeyboardEventsExample extends Application {
    public static void main(String[] args) {
        launch(args);
    }
    @Override
    public void start(Stage primaryStage) throws InterruptedException {

        Label label = new Label("This app will shut down automatically");
        VBox vbox = new VBox(label);

        Scene scene = new Scene(vbox);
        primaryStage.setScene(scene);
        primaryStage.setWidth(300);
        primaryStage.setHeight(100);
        primaryStage.show();

        primaryStage.addEventHandler(KeyEvent.KEY_PRESSED,  (event) -> {
            System.out.println("Key pressed: " + event.toString());
            switch(event.getCode().getCode()) {
                case 27 : { // 27 = ESC key
                    primaryStage.close();
                    break;
                }
                case 10 : { // 10 = Return
                    primaryStage.setWidth( primaryStage.getWidth() * 2);
                }
                default:  {
                    System.out.println("Unrecognized key");
                }
            }
        });
    }
}
```

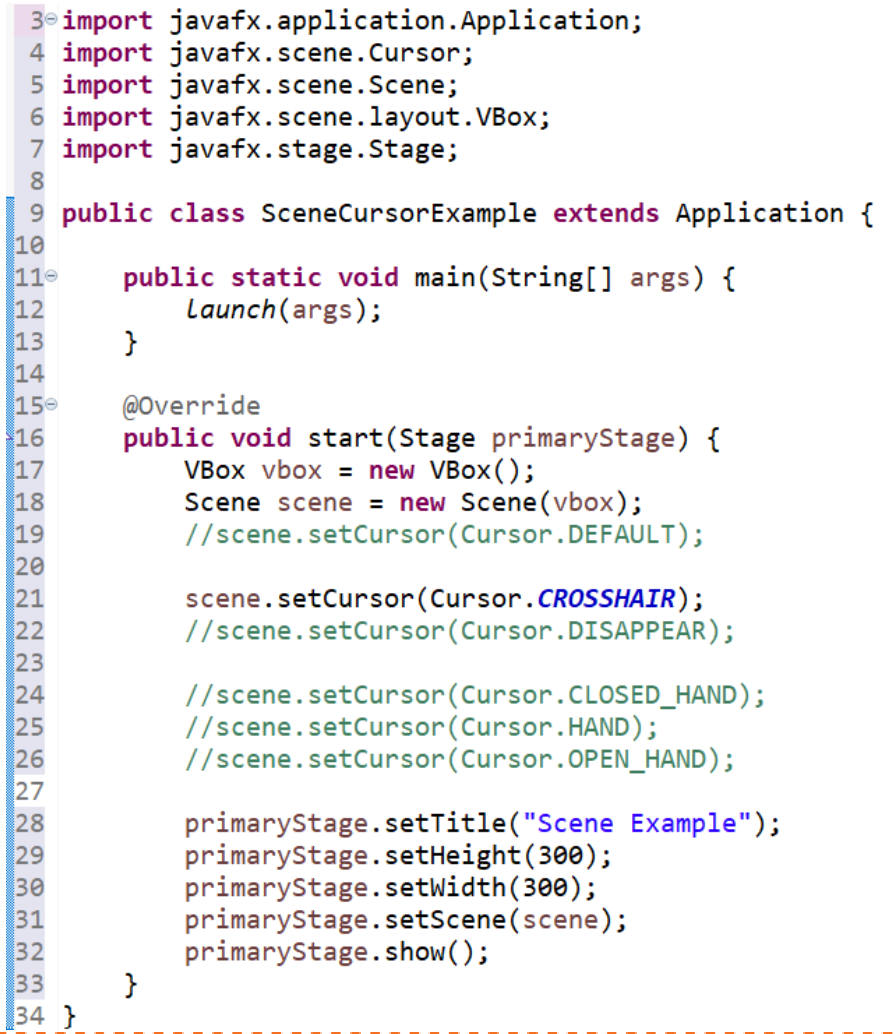
JavaFX Scene

- The JavaFX Scene object is the root of the JavaFX Scene graph.
- JavaFX Scene contains all the visual JavaFX GUI components inside it.
- A JavaFX Scene is represented by the class javafx.scene.Scene.
- A Scene object has to be set on a JavaFX Stage to be visible.

- Operations
  - Create Scene
  - Set Scene on Stage
  - The Scene Graph
  - Scene Mouse Cursor

JavaFX Scene

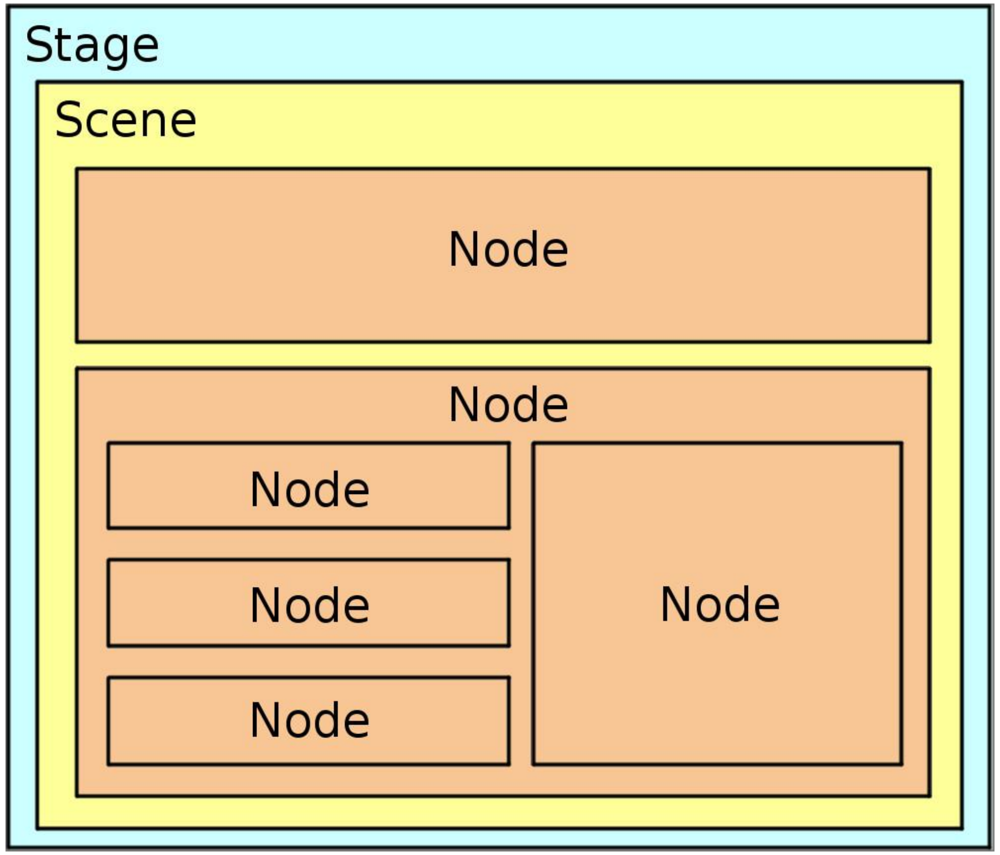
- The example shows how to set a specific mouse cursor



```
import javafx.application.Application;
import javafx.scene.Cursor;
import javafx.scene.Scene;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class SceneCursorExample extends Application {

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

    @Override
    public void start(Stage primaryStage) {
        VBox vbox = new VBox();
        Scene scene = new Scene(vbox);
        //scene.setCursor(Cursor.DEFAULT);

        scene.setCursor(Cursor.CROSSHAIR);
        //scene.setCursor(Cursor.DISAPPEAR);

        //scene.setCursor(Cursor.CLOSED_HAND);
        //scene.setCursor(Cursor.HAND);
        //scene.setCursor(Cursor.OPEN_HAND);

        primaryStage.setTitle("Scene Example");
        primaryStage.setHeight(300);
        primaryStage.setWidth(300);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

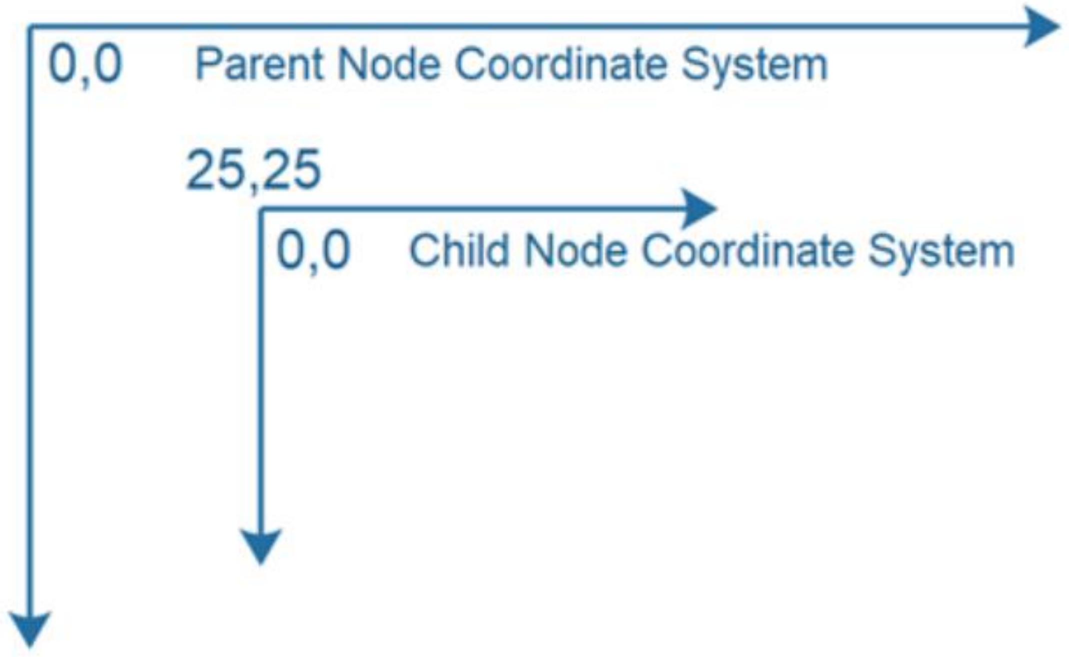
JavaFX Node

- Each JavaFX Node (subclass) instance can only be added to the JavaFX scene graph once.
- Node instance can only appear in one place in the scene graph.



JavaFX Node Properties

- A cartesian coordinate system
- A bounding box delimited by: Layout bounds, Bounds in local, Bounds in parent
- layoutX
- layoutY
- Preferred height
- Preferred width
- Minimum height
- Minimum width
- Maximum height
- Maximum width
- User data
- Items (Child nodes)



JavaFX Properties

- A JavaFX Property is a special kind member variable of JavaFX controls.

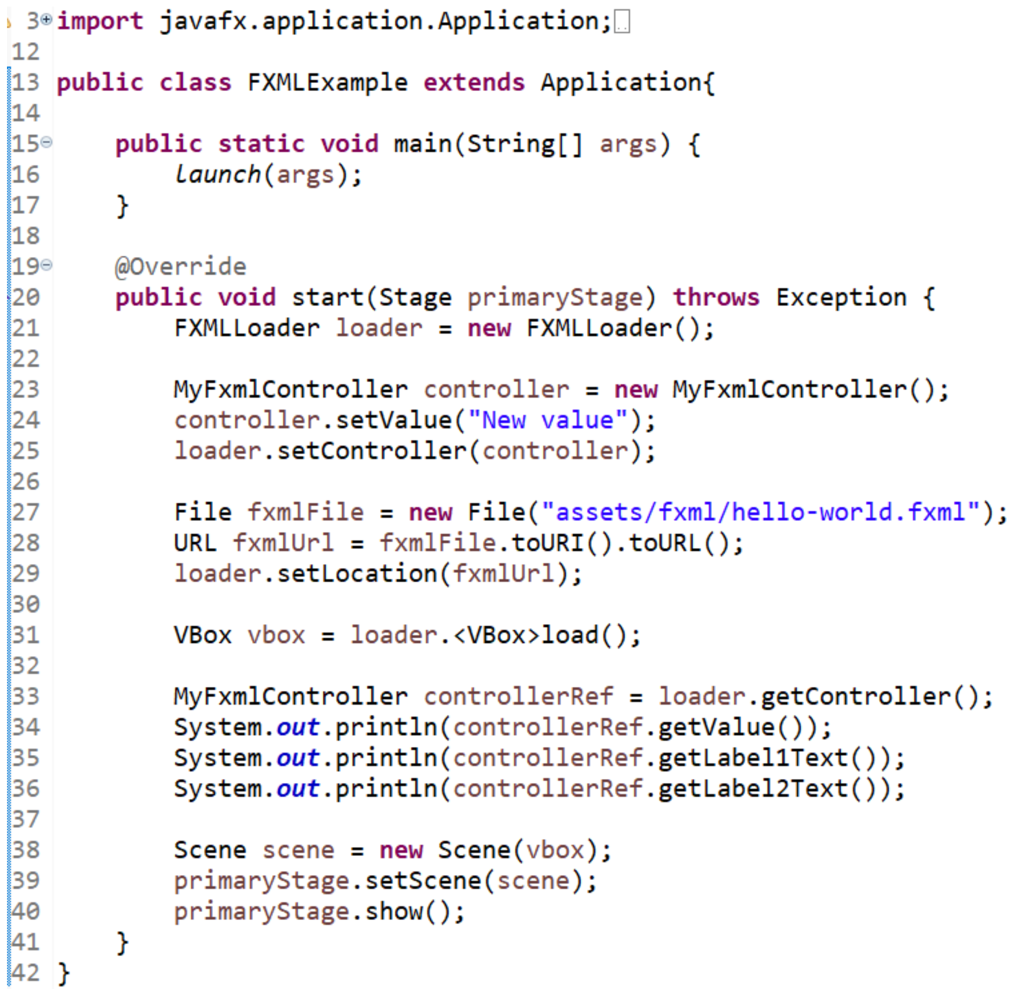
- JavaFX properties are typically used to contain control properties such as X and Y position, width and height, text, children and other central properties of JavaFX controls.
- Can attach change listeners to JavaFX properties so other components can get notified when the value of the property changes, and can bind properties to each other so when one property value changes, so does the other.

JavaFX FXML

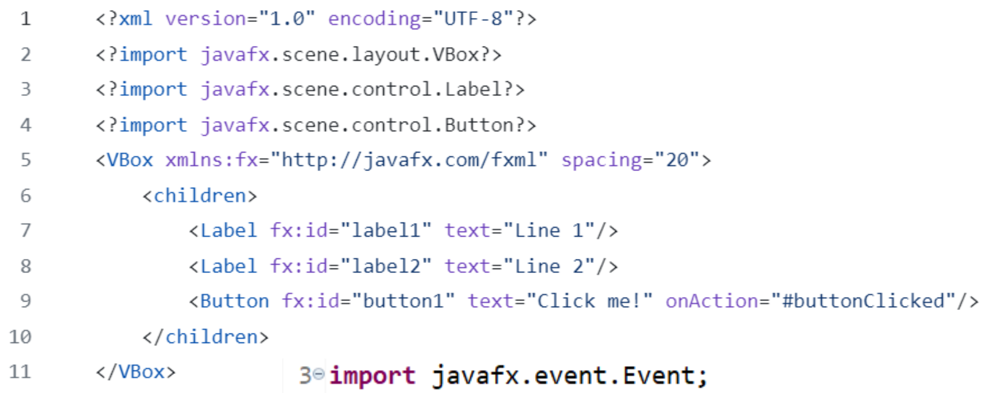
- JavaFX FXML is an XML format that enables you to compose JavaFX GUIs in a fashion similar to how you compose web GUIs in HTML.
- FXML enables to separate the JavaFX layout code from the rest of the application code. This cleans up both the layout code and the rest of the application code.

- FXML can be used both to compose the layout of a whole application GUI, or just part of an application GUI, e.g. the layout of one part of a form, tab, dialog etc.

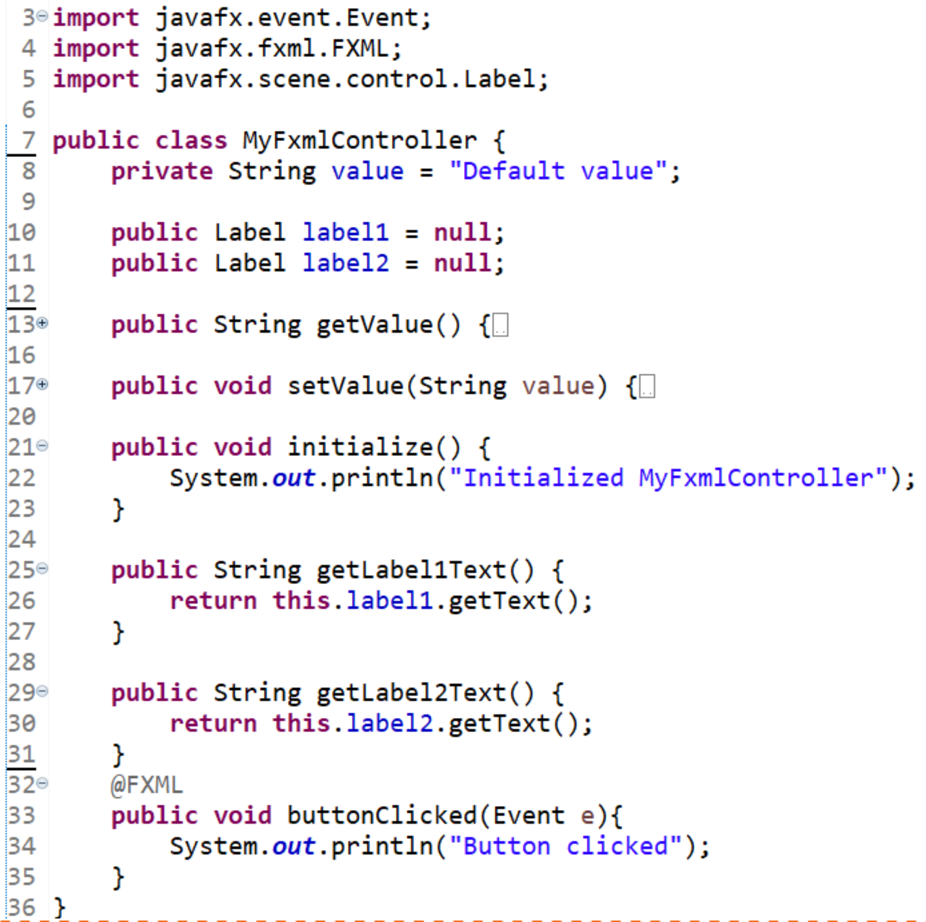
JavaFX FXML Example



```java
import javafx.application.Application;

public class FXMLExample extends Application{

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

    @Override
    public void start(Stage primaryStage) throws Exception {
        FXMLLoader loader = new FXMLLoader();

        MyFxmlController controller = new MyFxmlController();
        controller.setValue("New value");
        loader.setController(controller);

        File fxmlFile = new File("assets/fxml/hello-world.fxml");
        URL fxmlUrl = fxmlFile.toURI().toURL();
        loader.setLocation(fxmlUrl);

        VBox vbox = loader.<VBox>load();

        MyFxmlController controllerRef = loader.getController();
        System.out.println(controllerRef.getValue());
        System.out.println(controllerRef.getLabel1Text());
        System.out.println(controllerRef.getLabel2Text());

        Scene scene = new Scene(vbox);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```



```xml
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.Button?>
<VBox xmlns:fx="http://javafx.com/fxml" spacing="20">
    <children>
        <Label fx:id="label1" text="Line 1"/>
        <Label fx:id="label2" text="Line 2"/>
        <Button fx:id="button1" text="Click me!" onAction="#buttonClicked"/>
    </children>
</VBox>
```



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

public class MyFxmlController {
    private String value = "Default value";

    public Label label1 = null;
    public Label label2 = null;

    public String getValue() {

    public void setValue(String value) {

    public void initialize() {
        System.out.println("Initialized MyFxmlController");
    }

    public String getLabel1Text() {
        return this.label1.getText();
    }

    public String getLabel2Text() {
        return this.label2.getText();
    }
    @FXML
    public void buttonClicked(Event e){
        System.out.println("Button clicked");
    }
}
```

JavaFX Layout, Control Components

JavaFX Layout Components

- Layout components to help organize and structure the user interface
- JavaFX layouts are components which contains other components inside them. The layout component manages the layout of the components nested inside it.
- JavaFX layout components are also sometimes called parent components because they contain child components, and because layout components are subclasses of the JavaFX class **javafx.scene.Parent**.
- Common layout components:
  - Group
  - Region
  - Pane
  - **VBox** and **HBox**: Arranges its child nodes in a single vertical column or in a single horizontal row.

JavaFX Layout Components

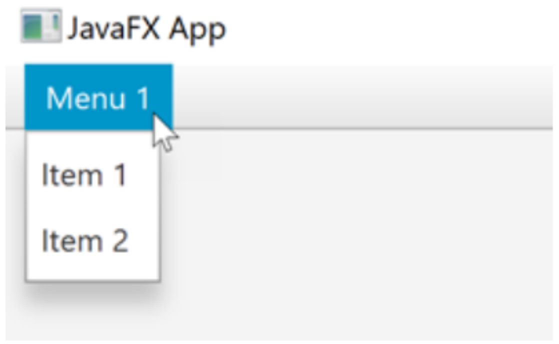
- **BorderPane**: Layout component divides the application window into five regions: top, bottom, left, right, and center.
- **FlowPane**: Arranges its child nodes in a flow that wraps at the boundary of the layout. It is useful for creating dynamically laid out content.
- **GridPane**: Allows for a flexible grid of rows and columns, where child nodes can be placed at specific row and column indices.
- **StackPane**: Layout component stacks its child nodes on top of each other.
- **AnchorPane**: Allows the positioning of nodes relative to the edges of the pane or relative to each other. It is useful for creating precise layouts.
- **TilePane**: Layout its children in a grid of equally sized cells.

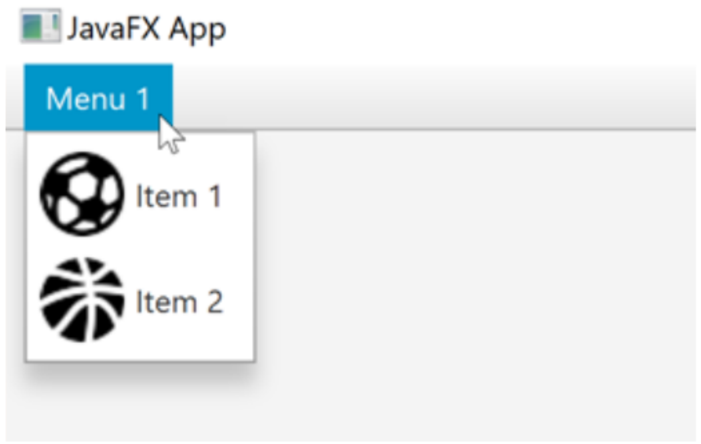
JavaFX Controls

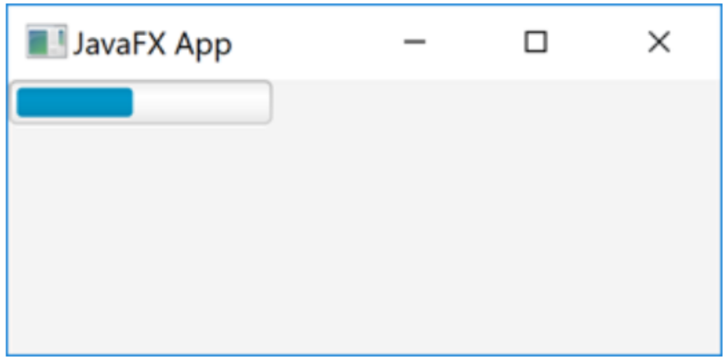
- JavaFX controls are JavaFX components which provide some kind of control functionality inside a JavaFX application.
- For a control to be visible it must be attached to the scene graph of some Scene object.
- Controls are usually nested inside some JavaFX layout component that manages the layout of controls relative to each other.

JavaFX Controls

- Common JavaFX Controls
  - Accordion, Button, CheckBox, ChoiceBox, ColorPicker,
  - ComboBox, DatePicker, Label, ListView, Menu, MenuBar, PasswordField
  - ProgressBar, RadioButton, Slider, Spinner, SplitMenuButton, TableView
  - TextArea, TextField, ToggleButton, ToolBar, TreeTableView, TreeView



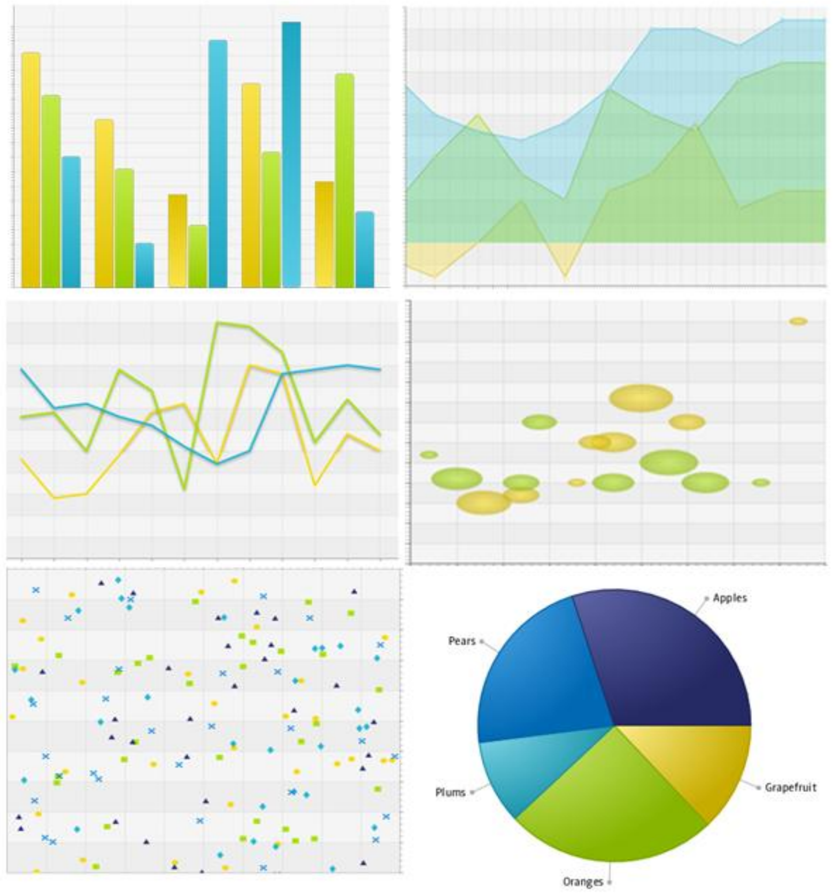




JavaFX Charts

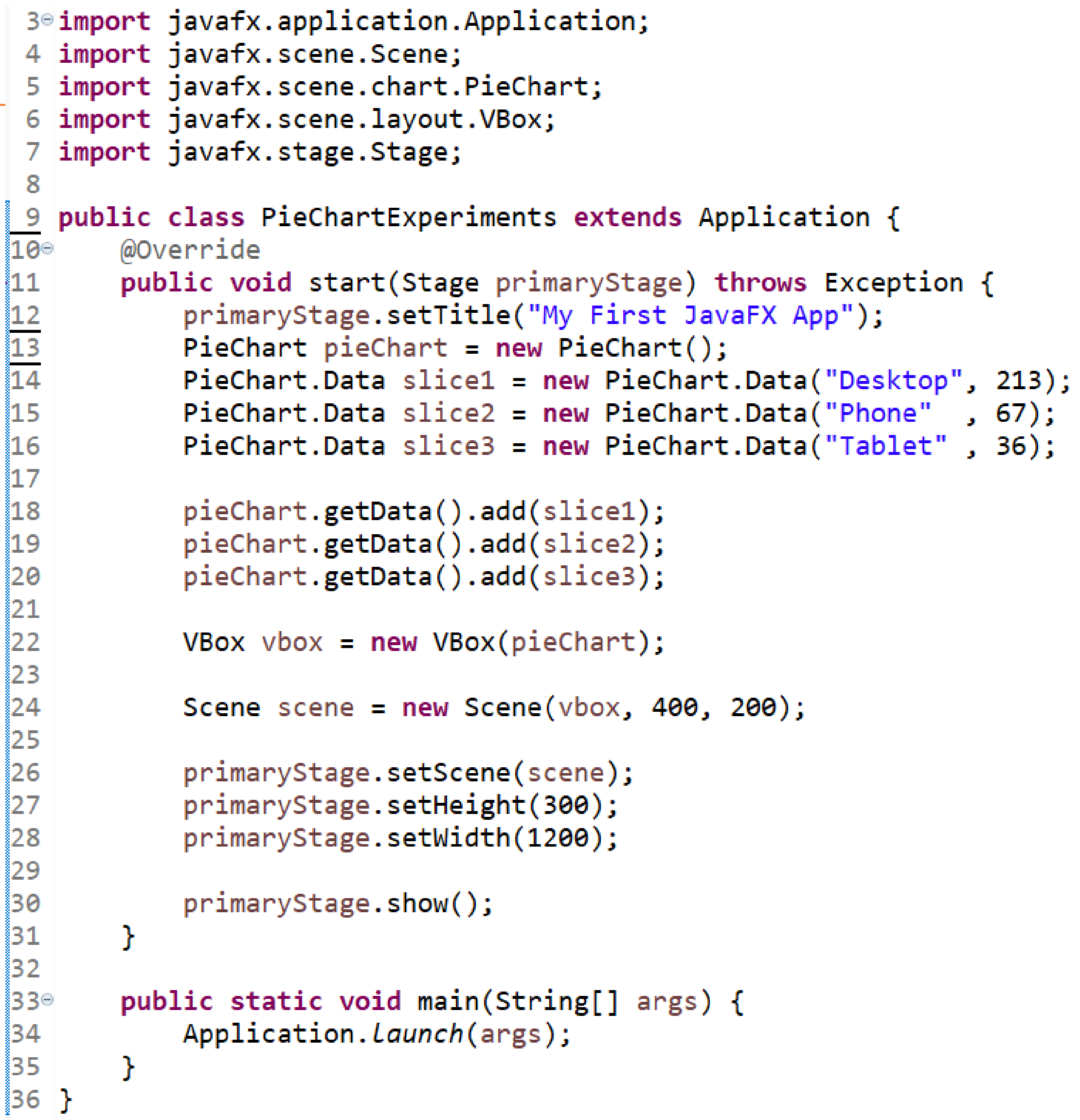
The JavaFX charts available in the **javafx.scene.chart** package.

- AreaChart
- BarChart
- BubbleChart
- LineChart
- PieChart
- ScatterChart
- StackedAreaChart
- StackedBarChart



JavaFX Charts

- The JavaFX PieChart component is capable of drawing pie charts in your JavaFX application based on data you supply it.
- The PieChart component is really easy to use.
- The JavaFX PieChart component is represented by the class **javafx.scene.chart.PieChart**.



```java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.chart.PieChart;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class PieChartExperiments extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("My First JavaFX App");
        PieChart pieChart = new PieChart();
        PieChart.Data slice1 = new PieChart.Data("Desktop", 213);
        PieChart.Data slice2 = new PieChart.Data("Phone"  , 67);
        PieChart.Data slice3 = new PieChart.Data("Tablet" , 36);

        pieChart.getData().add(slice1);
        pieChart.getData().add(slice2);
        pieChart.getData().add(slice3);

        VBox vbox = new VBox(pieChart);

        Scene scene = new Scene(vbox, 400, 200);

        primaryStage.setScene(scene);
        primaryStage.setHeight(300);
        primaryStage.setWidth(1200);

        primaryStage.show();
    }

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

JavaFX 2D Graphics

- JavaFX contains features that makes it easy to draw 2D graphics on the screen.
- 2D shape is represented by a class and all these classes belongs to the package **javafx.scene.shape**.
- Predefined shapes such as Line, Rectangle, Circle, Ellipse, Polygon, Polyline, Cubic Curve, Quad Curve, Arc.
- Path elements such as MoveTO Path Element, Line, Horizontal Line, Vertical Line, Cubic Curve, Quadratic Curve, Arc.
- In addition to these, you can also draw a 2D shape by parsing SVG path.

JavaFX 3D Graphics

- JavaFX contains features that makes it easy to draw 3D graphics on the screen.
- In general, a 3D shape is a geometrical figure that can be drawn on the XYZ plane.
- They are defined by two or more dimensions, commonly length, width and depth.
- 3D shapes supported by JavaFX include a Cylinder, Sphere and a Box.

- All 3D shape classes belong to the package **javafx.scene.shape.**

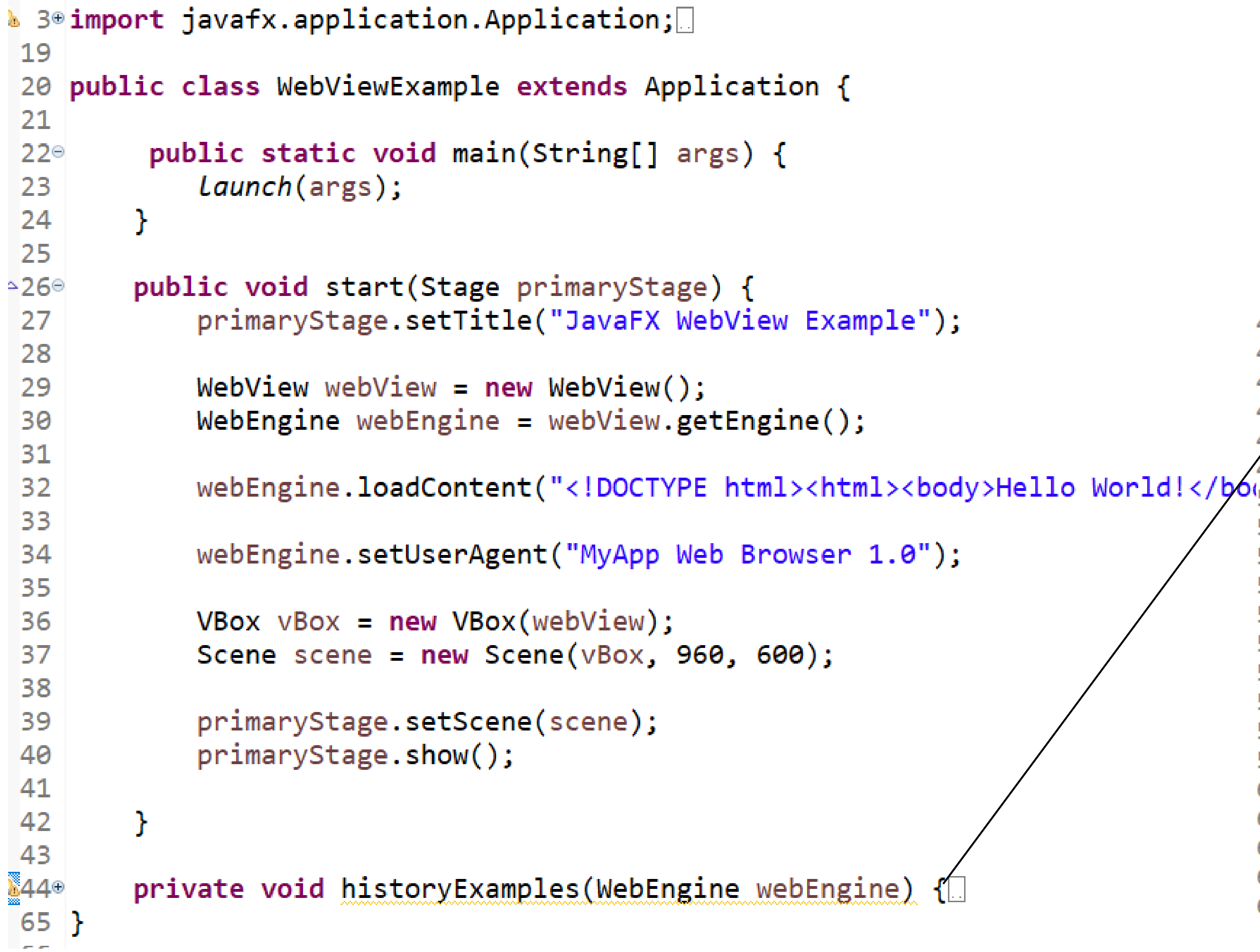
JavaFX Audio, Video

- JavaFX contains features that makes it easy to play audio, video in JavaFX applications.
- The package **javafx.scene.media** contains the classes and interfaces to provide media functionality in JavaFX.
- It is provided in the form of three components
  - Media Object − This represents a media file
  - Media Player − To play media content.
  - Media View − To display media.

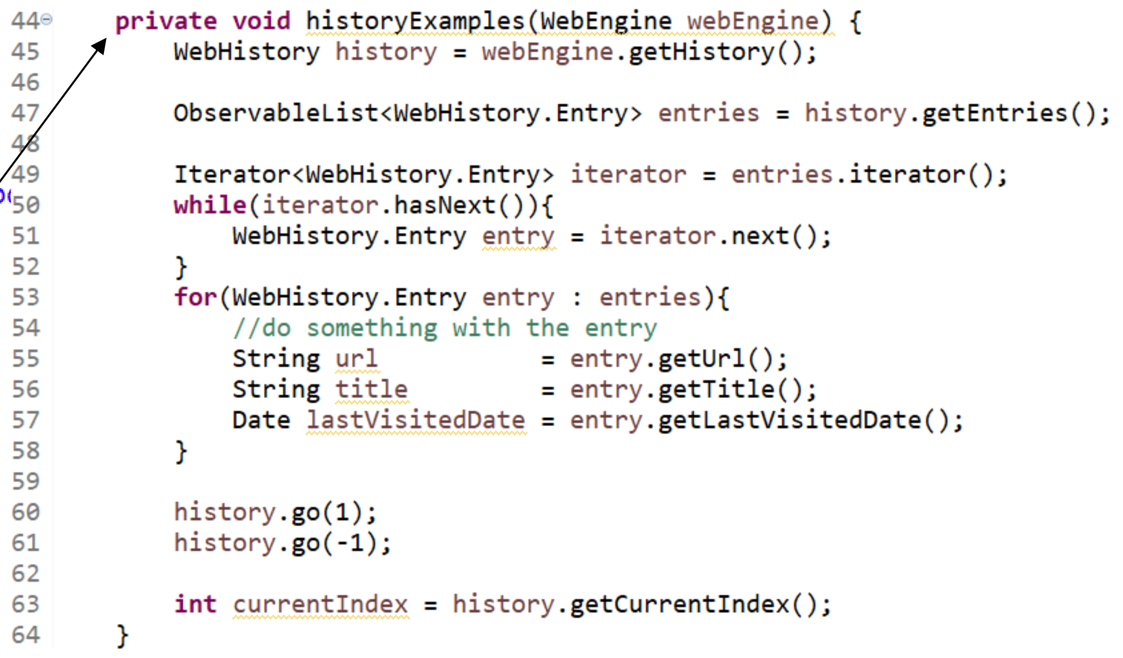
JavaFX WebView

- The JavaFX WebView (javafx.scene.web.WebView) component is capable of showing web pages (HTML, CSS, SVG, JavaScript) inside a JavaFX application. As such, the JavaFX WebView is a mini browser.
- The JavaFX WebView uses the WebKit open source browser engine internally to render the web pages.
- The JavaFX WebView WebEngine (javafx.scene.web.WebEngine) is an internal component used by the WebView to load the data that is to be displayed inside the WebView.
- To make the WebView WebEngine load data, you must first obtain the WebEngine instance from the WebView.

JavaFX WebView



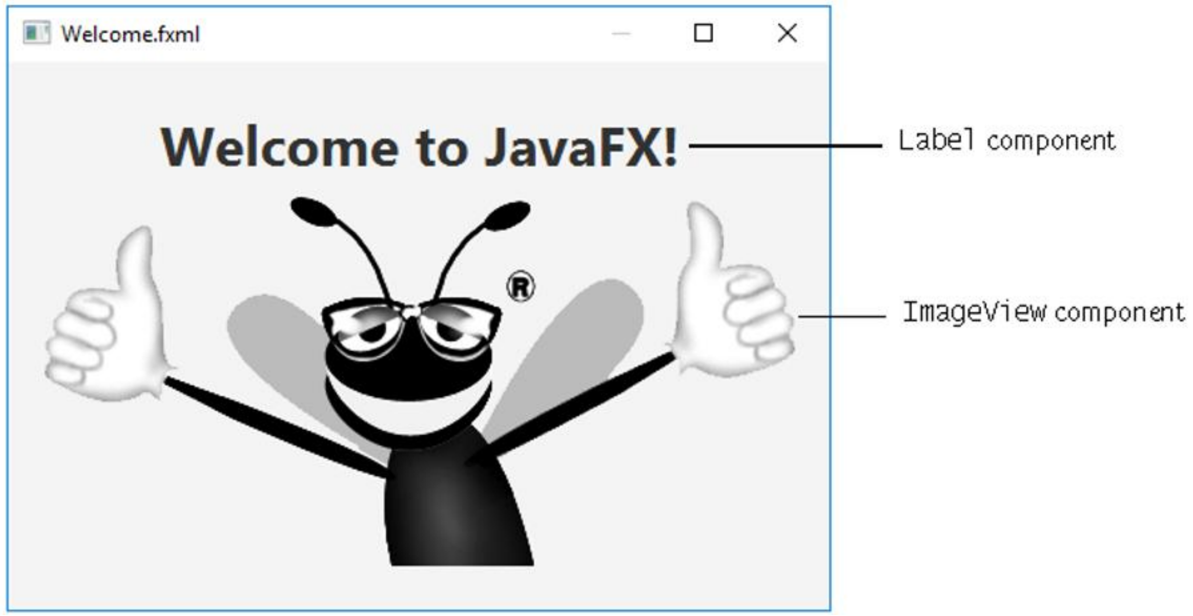
```java
import javafx.application.Application;

public class WebViewExample extends Application {

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

    public void start(Stage primaryStage) {
        primaryStage.setTitle("JavaFX WebView Example");

        WebView webView = new WebView();
        WebEngine webEngine = webView.getEngine();

        webEngine.loadContent("<!DOCTYPE html><html><body>Hello World!</bo

        webEngine.setUserAgent("MyApp Web Browser 1.0");

        VBox vBox = new VBox(webView);
        Scene scene = new Scene(vBox, 960, 600);

        primaryStage.setScene(scene);
        primaryStage.show();

    }

    private void historyExamples(WebEngine webEngine) {
}
```



```java
    private void historyExamples(WebEngine webEngine) {
        WebHistory history = webEngine.getHistory();

        ObservableList<WebHistory.Entry> entries = history.getEntries();

        Iterator<WebHistory.Entry> iterator = entries.iterator();
        while(iterator.hasNext()){
            WebHistory.Entry entry = iterator.next();
        }
        for(WebHistory.Entry entry : entries){
            //do something with the entry
            String url           = entry.getUrl();
            String title         = entry.getTitle();
            Date lastVisitedDate = entry.getLastVisitedDate();
        }

        history.go(1);
        history.go(-1);

        int currentIndex = history.getCurrentIndex();
    }
```

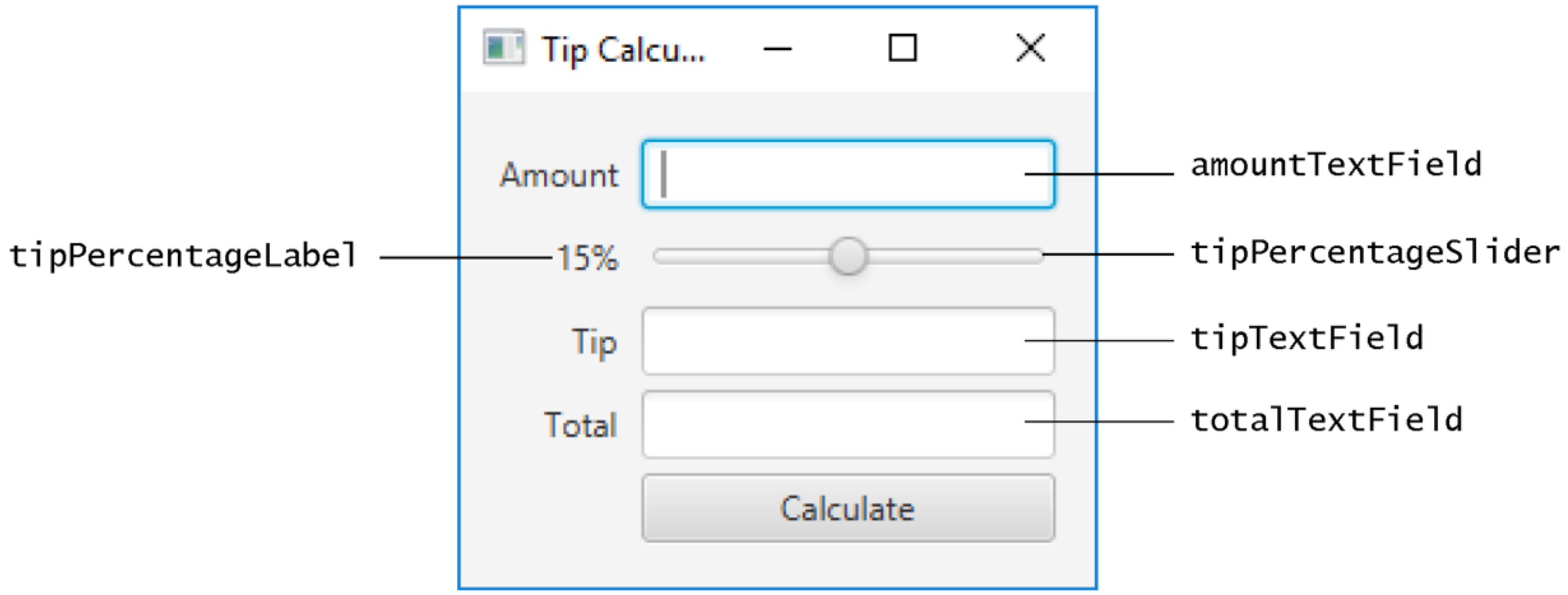
JavaFX Demonstration

Demonstration JavaFX Simple Application


Welcome.fxml

Welcome to JavaFX!

Demonstration JavaFX Simple Application



JavaFX Concurrency

- *JavaFX concurrency* refers to how JavaFX is designed with respect to multithreading and concurrency.
- JavaFX uses a single-threaded rendering design, meaning only a single thread can render anything on the screen, and that is the JavaFX application thread. In fact, only the JavaFX application thread is allowed to make any changes to the JavaFX Scene Graph in general.
- A single-threaded rendering design is easier to implement correctly, but long-running tasks that run within the JavaFX application thread make the GUI unresponsive until the task is completed. No JavaFX GUI controls react to mouse clicks, mouse over, keyboard input while the JavaFX application thread is busy running that task.

Summary

Concepts were introduced:

- JavaFX Architecture
- JavaFX Core: Stage, Scene, FXML
- Understand JavaFX Properties and Bindings, Events, Layouts, Controls
- Create Java desktop applications with JavaFX