JavaFX

ORACLE

Objectives

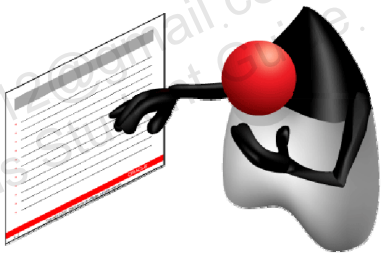
After completing this lesson, you should be able to:

- Describe the features of JavaFX
- Identify the features of the JavaFX scene graph
- Describe the JavaFX development tools
- Describe how JavaFX is integrated into a Java application
- Use FXML and determine when to use it in an application
- Download and install JavaFX and related samples

ORACLE

Topics

- Describe the features of JavaFX
- Describe a JavaFX application
- Download and install JavaFX and related samples



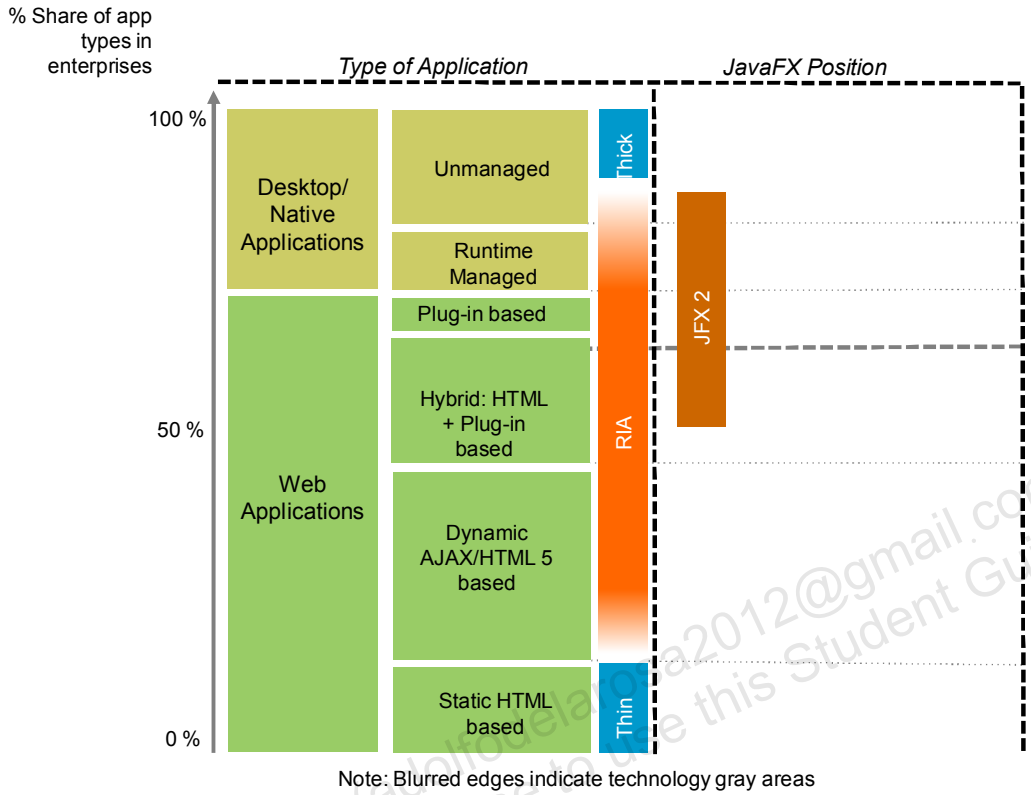
JavaFX Features

- Integrated with Java SE 7
- More than 50 charts, forms, and layout components
- Platform includes FXML, which is an alternate language to create a UI.
- Integrates JavaFX into Swing applications
- Improved graphics engine
- Integrates with scripting languages such as Groovy, JRuby, and Scala

ORACLE

- Java APIs for JavaFX: JavaFX 2 applications are completely developed in Java.
- JavaFX is integrated into existing Java libraries.
- Use your favorite Java development tools.
- Use popular JVM-based scripting languages such as Groovy, JRuby, and Scala.
- Develop and maintain complex user interfaces easily.
- Web-rendering engine
- Mix and match native Java capabilities and the dynamic capabilities of web technologies in your applications.
- High-performance, hardware-accelerated graphics pipeline
- High-performance media engine
- Play back video and audio content in popular formats within your application.

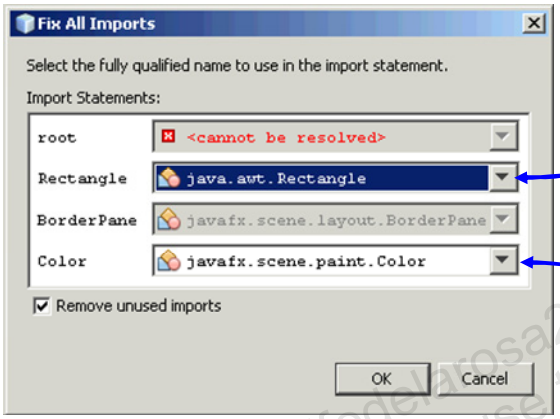
Overview of JavaFX in the Enterprise



In the realm of enterprise application, JavaFX is a rich client that sits between a web application and a desktop application. Other tools that could fit in this space include Flash, Flex, Oracle ADF, GWT, Silverlight, and others.

Swing

- Swing has been updated in Java SE 7
- You can mix Swing and JavaFX in an application (although we do not cover such mixing in this course).
- Do not confuse Swing, AWT, and JavaFX APIs.



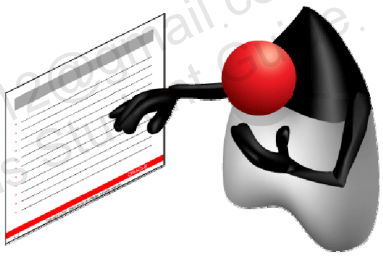
Swing is still available and supported in Java SE 7. As you develop JavaFX applications, you have to choose which API to use when you create objects. NetBeans enables you to "Fix Imports" instead of typing the package statements. The image in the slide shows that you are given choices that include JavaFX and AWT. We use JavaFX throughout the course, so be sure to choose the JavaFX packages.

Swing is fully supported in JDK 7 and has the following enhancements, including:

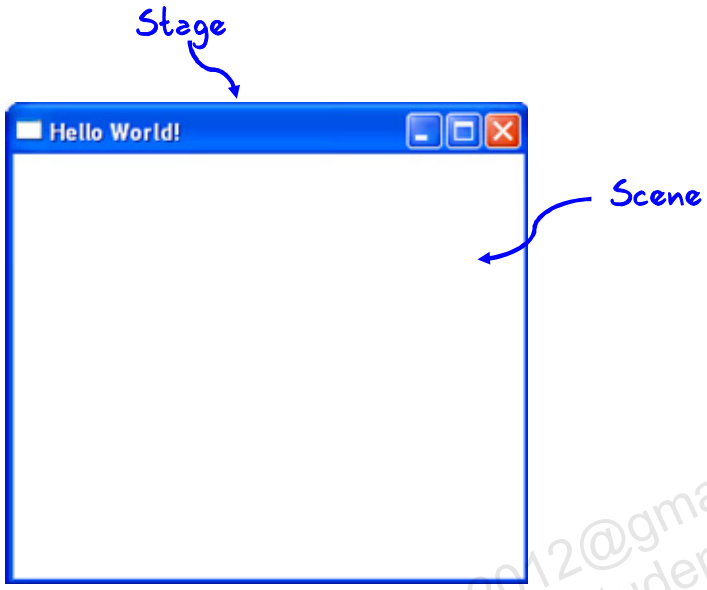
- JLayer class
- Nimbus Look and Feel
- Heavyweight and Lightweight components
- Shaped and translucent windows
- Hue-Saturation-Luminance (HSL) Color Selection in JColorChooser Class

Topics

- Describe the features of JavaFX
- Describe a JavaFX application
- Download and install JavaFX and related samples



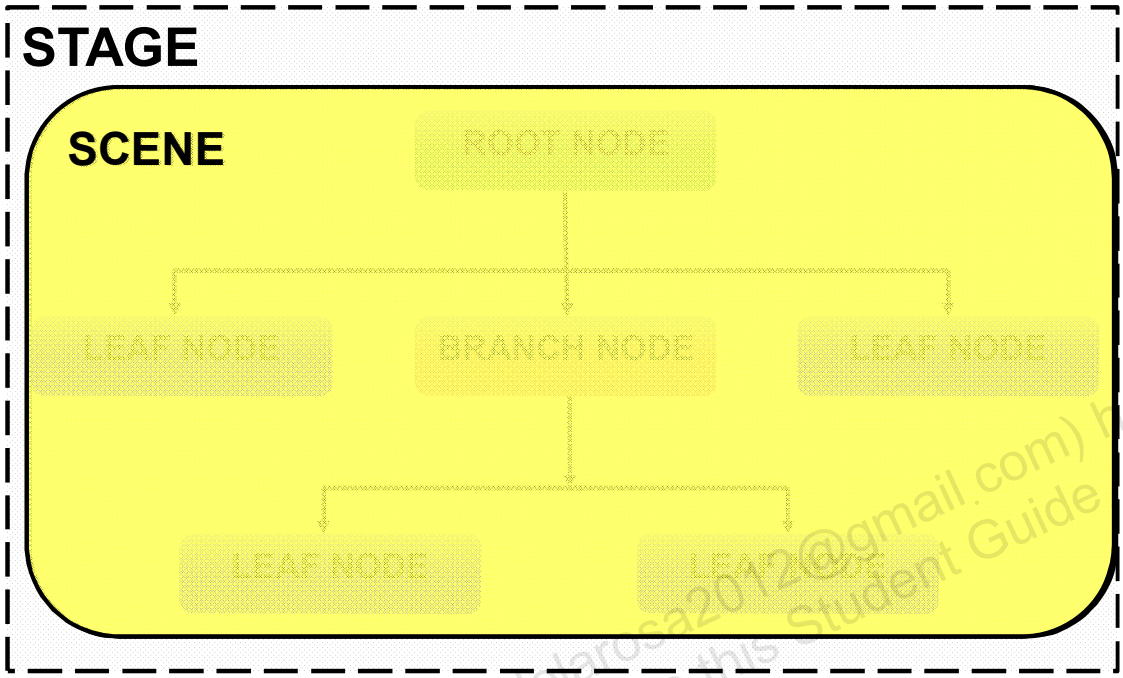
Stage and Scene



The image is an example of an application with a stage and a scene. Even applications embedded in a browser have a stage in the application code. A stage can be decorated with a window title or not, but there is always a stage.

Notice that the stage in the example is constructed by the platform, which in this case is a Windows XP platform.

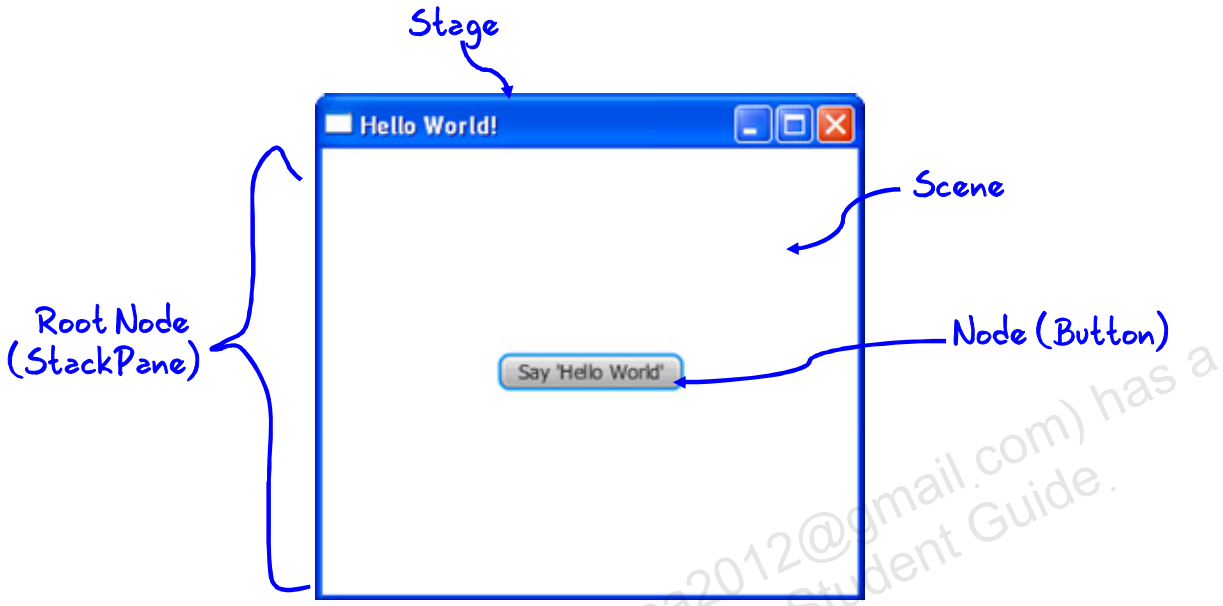
JavaFX Scene Graph: Stage and Scene



The stage (`javafx.stage.Stage`) is the top-level GUI container for all graphical objects, and the scene is the base container. The primary stage is constructed by the platform. Additional stage objects may be constructed by the application. Stage objects must be constructed and modified on the JavaFX Application Thread.

The JavaFX `Scene` class (`javafx.scene.Scene`) is the container for all content in a scene graph.

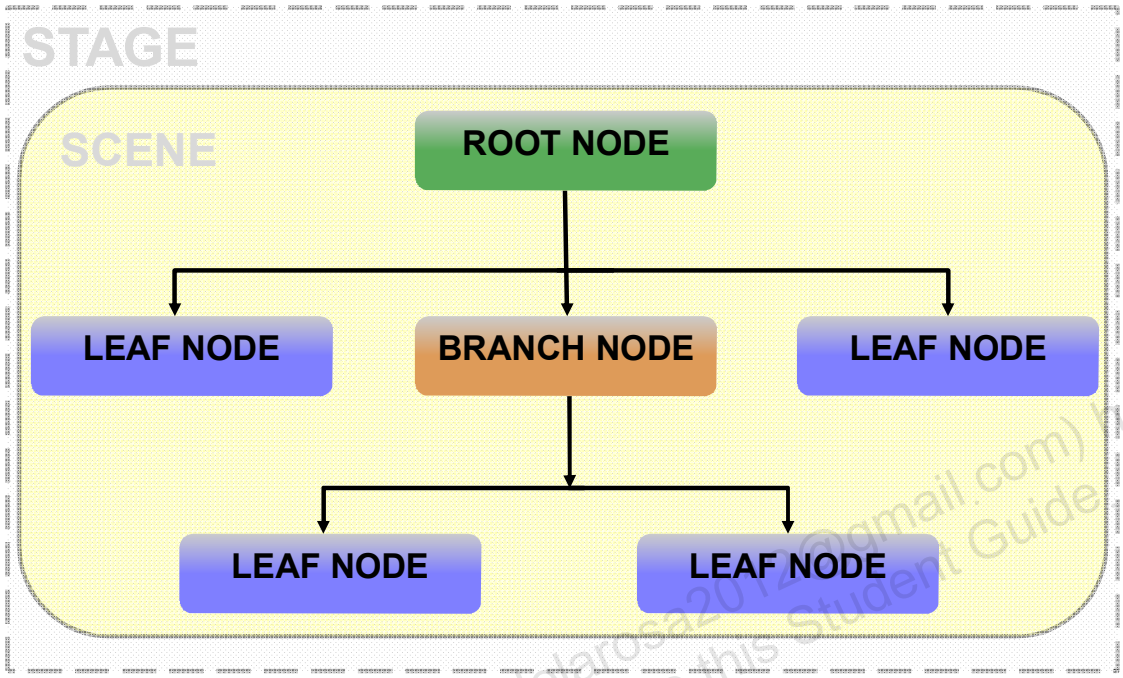
Stage, Scene, and Nodes



Here is an example of an application with a stage, a scene, and some nodes. The nodes in the example include a StackPane and a button. A StackPane is a layout container, and a button is a component.

These are covered in the lesson titled "*Visual Effects, Animation, Web View, and Media*."

JavaFX Scene Graph: Nodes Hierarchy

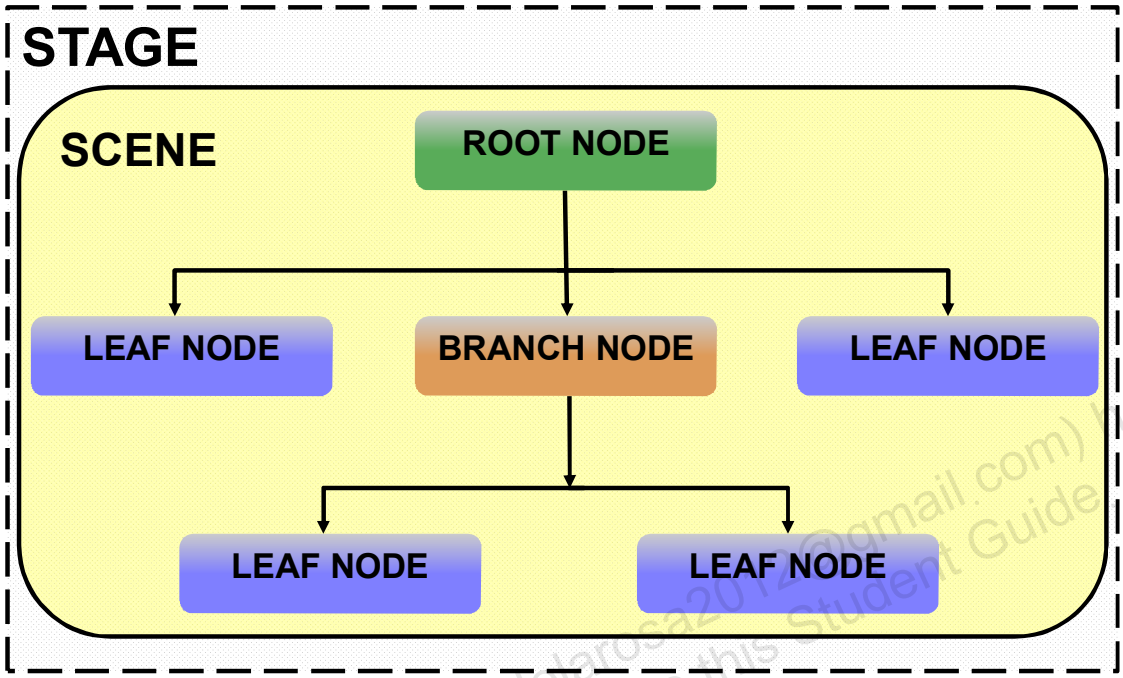


The individual items held within the JavaFX scene graph are known as *nodes*. Each node is classified as one of the following:

- A branch node (or parent, meaning that it can have children)
- A leaf node

The top node in the tree is called the *root node*, and it does not have a parent.

JavaFX Scene Graph



A scene graph is a tree data structure, most commonly found in graphical applications and libraries such as vector editing tools, 3D libraries, and video games.

The JavaFX scene graph API makes graphical user interfaces easier to create, especially when complex visual effects and transformations are involved.

The JavaFX scene graph is a retained-mode API, meaning that it maintains an internal model of all graphical objects in your application. At any given time, it knows what objects to display, what areas of the screen need repainting, and how to render it all in the most efficient manner. Instead of invoking primitive drawing methods directly, you use the scene graph API and let the system automatically handle the rendering details. This approach significantly reduces the amount of code that is needed in your application.

Rich Client Application

The image in the slide is the Data App sample, which is a rich client developed using JavaFX. The labels show the various parts of the scene graph of the application.

The scene is the main window of the application, and each tab is displayed in that scene when it is clicked.

Each tab can be considered to be a scene graph. On each tab, there are nodes such as charts, tables, labels, buttons, and more.

Minimum JavaFX Application

Basic Java application with enabled JavaFX features

```
1  package javafxapplication;

                                        Code creates a stage only.

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

3  public class JavaFXApplication extends Application {


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

5     @Override public void start(Stage primaryStage) {

6         primaryStage.show();
      }

   }
```

ORACLE

A JavaFX application is a basic Java application with enabled JavaFX features. The minimum code needed to run a JavaFX application consists of the following:

- An extension of the `javafx.application.Application` class
- A `main()` method that calls the `launch()` method an override of the `start()` method
- A primary stage that is visible

The following refers to the code sample:

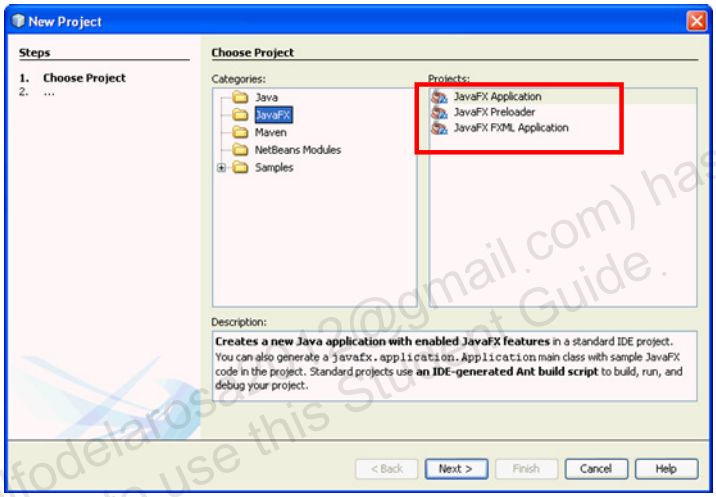
1. Package name
2. Package imports for `javafx.stage.Stage` and `javafx.application.Application`
3. The class you create inherits all the features and functionality of the `Application` class.
4. `main()` method, which calls the `launch()` method. As a best practice, the `launch()` method is the only method that is called by the `main()` method.
5. The `JavaFXApplication` class overrides the abstract `start()` method in the `Application` class. The `start()` method takes as an argument the primary stage for the application.
6. The final line of code makes the stage visible.

If you run this code, you see a stage without a scene.

JavaFX Application Types

Three types of JavaFX applications:

- JavaFX
- JavaFX FXML
- JavaFX preloader



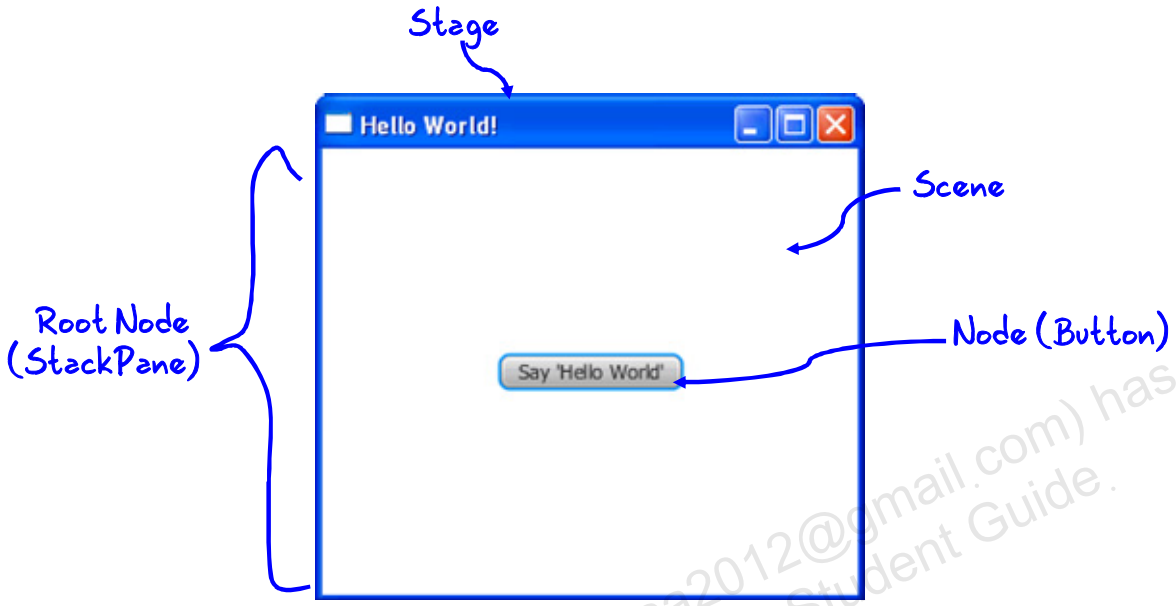
When you create a new JavaFX application by using NetBeans, you have a choice of three options. Each option creates a simple, prebuilt application.

JavaFX: Uses traditional Java syntax with JavaFX APIs

FXML: FXML is new for JavaFX 2. FXML is an XML-based declarative markup language for defining the user interface in a JavaFX application. FXML is well suited for defining static layout such as forms, controls, and tables. With FXML, you can also construct layouts dynamically by including a script.

JavaFX preloader: Used in deployment

Simple JavaFX Application



The JavaFX and FXML code examples in the next several slides represent the application shown in this slide. The application includes a stage, a scene, a button with text, and a layout container.

JavaFX Java Class

```
public class JavaFXApplication extends Application {

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

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Hello World!");
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.setOnAction(new EventHandler<ActionEvent>() {

            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });

        StackPane root = new StackPane();
        root.getChildren().add(btn);
        primaryStage.setScene(new Scene(root, 300, 250));
        primaryStage.show();
    }
}
```

Create Stage

Create Button (leaf node)

Create StackPane (root node)

Set the Scene

ORACLE

The code shown in the example is a default JavaFX application that you create in the NetBeans IDE. This example creates the application shown in slide 22.

The next three slides are the FXML equivalent of the same application.

JavaFX FXML

FXML is an XML-based declarative markup language

- Defines static layout such as forms, controls, and tables
- Constructs layouts dynamically by including a script

```
<AnchorPane>
    <children>
        <Button/>
        <Label/>
    </children>
</AnchorPane>
```

Group Node

Child Nodes

FXML offers the following advantages:

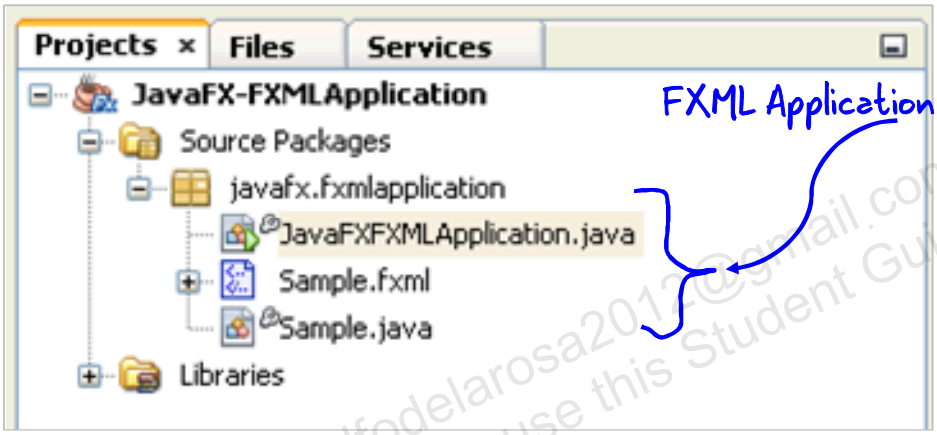
- FXML is based on XML and is familiar to most developers, especially web developers and developers using other RIA platforms.
- FXML is not a compiled language. You do not need to recompile the code to see the changes you make.
- FXML makes it easy to see the structure of your application's scene graph because of its hierarchical structure. This, in turn, makes it easier for members of a development team to collaborate on the application.

Note: The `<children>` and `</children>` tags are optional, but you can use them to improve the readability of your code.

JavaFX FXML Application Format

Three required files:

- *Applicationclass*.java
- Sample.java
- Sample.fxml



When you create a new JavaFX FXML application by using NetBeans, three files are created:

- ***Applicationclass*.java**: Contains code for creating the application scene and stage and for launching the application. The following line is especially important to understand:

```
Parent root =
FXMLLoader.load(getClass().getResource("Sample.fxml"));
```

  The `FXMLLoader.load()` method loads the object hierarchy from the resource file `Sample.fxml` and assigns it to the variable named root.
- **Sample.fxml**: This file is where you build the user interface. The NetBeans IDE automatically populates this file with markup for an Anchor Pane layout, which includes Button and Label components.
- **Sample.java**: This file is the controller file. NetBeans automatically populates this file so that it defines the action event for the button in the `Sample.fxml` file.

JavaFX FXML Application Format:
JavaFXFXMLApplication.java

```
public class JavaFXFXMLApplication extends Application {

    public static void main(String[] args) {
        Application.launch(JavaFXFXMLApplication.class,
    args);
    }
    @Override
    public void start(Stage stage) throws Exception {
        Parent root =
    FXMLLoader.load(getClass().getResource("Sample.fxml")
    );
        stage.setScene(new Scene(root));
        stage.show();
    }
}
```

Create Stage

Loads contents of Scene and makes Sample.fxml root

Set the Scene

In this example, the `JavaFXFXMLApplication` class is created. This is one of three classes required for a JavaFX FXML application.

By default, NetBeans generates code that creates the scene and stage and launches the application. This class functions as the application's main class.

The `FXMLLoader.load()` method loads the object hierarchy from the resource file `Sample.fxml` and assigns it to the variable named root.

This class is the Model in the MVC pattern.

JavaFX FXML Application Format: `Sample.fxml`

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

<?import java.lang.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<AnchorPane id="AnchorPane" prefHeight="200" prefWidth="320"
xmlns:fx=http://javafx.com/fxml
fx:controller="javafx.fxmlapplication.Sample">
    <children>
        <Button id="button" layoutX="126" layoutY="90"
         text="Click Me!" onAction="#handleButtonAction"
         fx:id="button" />
        <Label id="label" layoutX="126" layoutY="120"
         minHeight="16" minWidth="69" prefHeight="16"
         prefWidth="69" fx:id="label" />
    </children>
</AnchorPane>
```

AnchorPane is root node

Child Nodes

The `Sample.fxml` file holds the content that sits in the stage and scene and functions as the application interface. By default, NetBeans creates a button with an action and a label. Typically, you will delete this automatically generated code and replace it with your own code. In addition, you will rename the file using an intuitive name that describes its function within the application. In most applications, this file will become the user interface. The class is the View in the MVC pattern.

JavaFX FXML Application Format: `Sample.java`

```
public class Sample implements Initializable {

    @FXML
    private Label label;

    @FXML
    private void handleButtonAction(ActionEvent event) {
        System.out.println("You clicked me!");
        label.setText("Hello World!");
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }
}
```

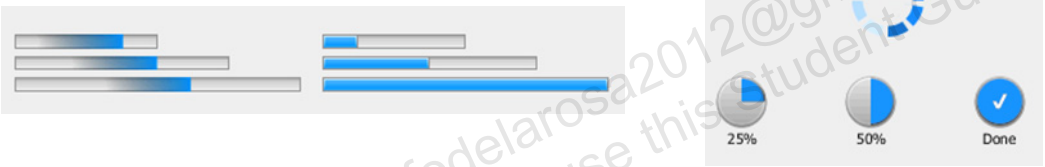
Declares this class as Controller class

Button event

By default, NetBeans creates a class that acts as the Controller class. Typically, you will delete this automatically generated code and replace it with your own code. In addition, you will rename the file using an intuitive name that describes its function within the application. This file will control `Sample.fxml`, which is the user interface. The name of this class should match the name of the fxml view class. This class acts as the Controller in the MVC pattern.

JavaFX Preloader

- A preloader is a small application that is started before the main application to customize the startup experience.
- JavaFX has two types of preloaders:
  - Default (exists in the runtime)
  - Custom (created by you)
- A preloader extends the `javafx.application.Preloader` package and can use a progress bar or progress indicator.



Preloaders improve the application startup experience. Use of a preloader can help to reduce perceived application startup time by showing some content to the user earlier, such as a progress indicator or login prompt.

A preloader application can also be used to present custom messaging to the user. For example, you can explain what is currently happening and what the user will be asked to do next, such as grant permissions to the application. Or you can create a preloader to present custom error messaging.

Not every application needs a preloader. For example, if the size of your application is small and does not have special requirements such as permissions, then it probably starts quickly. Even for larger applications, the default preloader included with the JavaFX Runtime can be a good choice, because it is loaded from the client machine rather than the network.

Preloaders are described in the lesson titled "*Packaging and Deploying Applications*."

Quiz

As a standard practice, which of the following is the only method called by the `main()` method?

a. `start()`
b. `initialize()`
c. `launch()`
d. `stop()`

Answer: c

a. The `main` class overrides the abstract `start()` method and calls the `start()` method.
b. The `initialize()` method is not called in the `main` class.
c. The `launch()` method is typically called by the `main()` method.
d. The `stop()` method is not called in the `main()` method.

Quiz

Which node in the JavaFX Scene Graph is always the top node?

a. Top
b. Simple
c. Graphic
d. Root

ORACLE

Answer: d

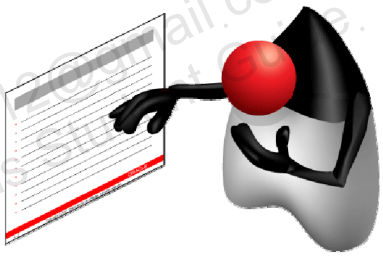
a. A Top node is not a type of node.
b. A Simple node is not a type of node.
c. Graphic is not a type of node.
d. Root nodes are the top node in the scene graph.

Practice 3: Overview

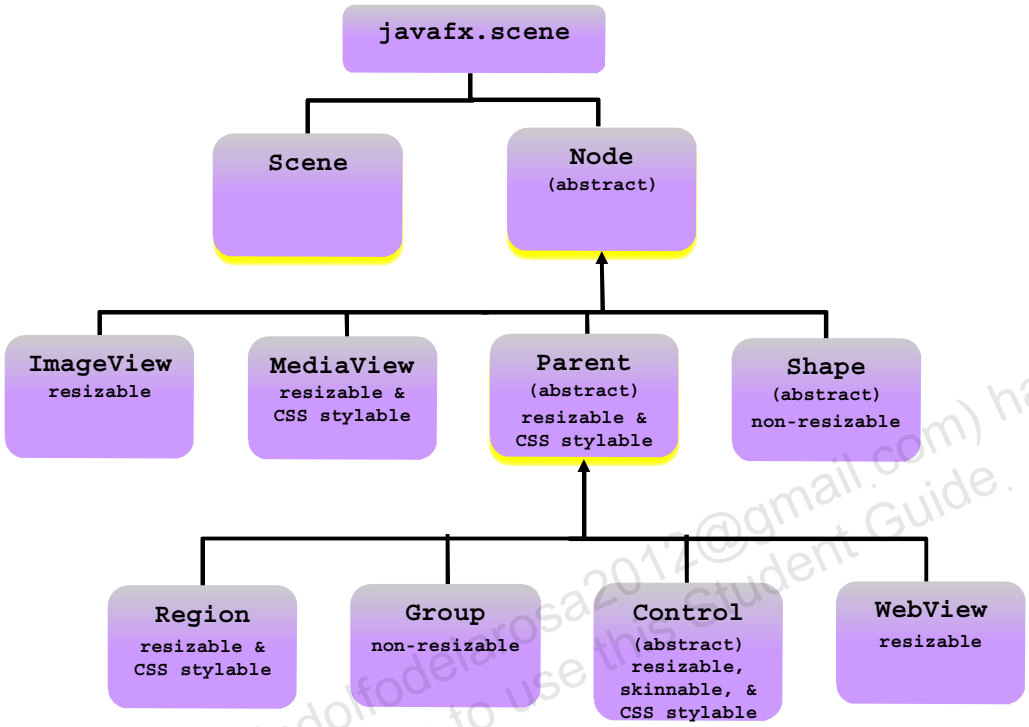
- 3-1: Creating a Login Window by Using JavaFX
- 3-2: Creating an FXML Login Window

Topics

- Describe the features of JavaFX
- Describe a JavaFX application
- Download and install JavaFX and related samples



Scene Graph API



The `javafx.scene` package defines more than a dozen classes, but three classes in particular are most important when it comes to learning how the API is structured.

- `Node`: The abstract base class for all scene graph nodes
- `Parent`: The abstract base class for all branch nodes (This class directly extends `Node`.)
- `Scene`: The base container class for all content in the scene graph

These base classes define important functionality that is subsequently inherited by subclasses, including paint order, visibility, composition of transformations, support for CSS styling, and so on.

`Parent` is the base class for all nodes that have children in the scene graph and is unique to JavaFX. This class handles all hierarchical scene graph operations, including adding and removing child nodes, marking branches as dirty for layout and rendering, picking, bounds calculations, and executing the layout pass on each pulse.

There are three direct concrete `Parent` subclasses:

- Group effects and transforms to be applied to a collection of child nodes
- `Region` class for nodes that can be styled with CSS and layout children
- Base `Control` class for high-level skinnable nodes designed for user interaction

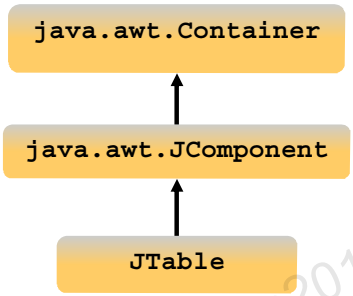
The JavaFX Scene class `javafx.scene.Scene` is the container for all content in a scene graph. The background of the scene is filled as specified by the fill property. The application must specify the root node for the scene graph by setting the root property. If `Group` is used as the root, the contents of the scene graph will be clipped by the scene's width and height and changes to the scene's size (if the user resizes the stage) will not alter the layout of the scene graph. If a resizable node (layout `Region` or `Control)` is set as the root, the root's size will track the scene's size, causing the contents to be relayed out as necessary.

The scene's size can be initialized by the application during construction. If no size is specified, the scene automatically computes its initial size based on the preferred size of its content.

Swing Containment Hierarchy

Levels of the Swing component hierarchy include:

- Frame
- Panel
- Component



The levels of the Swing containment hierarchy are similar to the JavaFX scene graph's hierarchy.

- The frame is a top-level container. It exists mainly to provide a place for other Swing components to paint themselves. The other commonly used top-level containers are dialogs (JDialog) and applets (JApplet).
- The panel is an intermediate container. Its only purpose is to simplify the positioning of the button and label. Other intermediate Swing containers, such as scroll panes (JScrollPane) and tabbed panes (JTabbedPane), typically play a more visible, interactive role in a program's GUI.
- The button and label are atomic components that exist not to hold random Swing components but as self-sufficient entities that present bits of information to the user. Atomic components also often get input from the user. The Swing API provides many atomic components, including combo boxes (JComboBox), text fields (JTextField), and tables (JTable).

Download and Install JavaFX

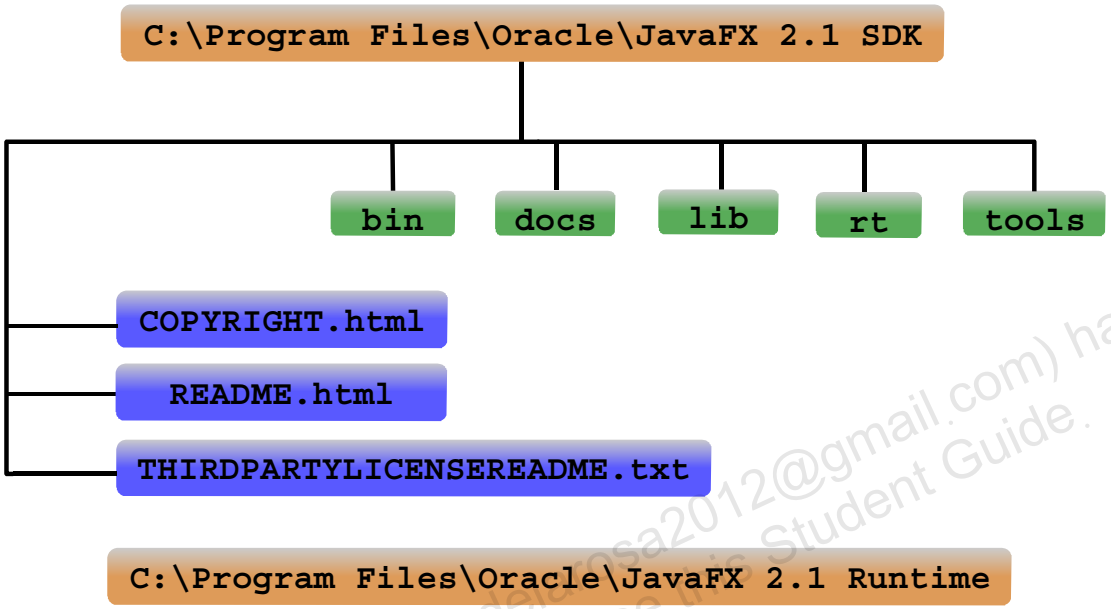
There are five options for downloading JavaFX bundles from oracle.com/javafx:

- JDK plus JavaFX SDK (includes Runtime)
- JavaFX SDK (includes Runtime)
- JavaFX Runtime only
- NetBeans with JavaFX
- Scene Builder: FXML editor

From the JavaFX downloads page, you can see there are several options for downloading JavaFX.

- The first option is to download JavaFX along with JDK 7. This option includes the JavaFX SDK and Runtime.
- The second option is the JavaFX SDK and JavaFX Runtime. The stand-alone JavaFX 2 SDK is recommended for developers using JDK 6u26 through 7u1. The JavaFX SDK includes the JavaFX libraries, the JavaFX Runtime, and project files that you need to get started. You can use the JavaFX SDK by itself or with your favorite Java IDE to leverage debugging, profiling, and other IDE features. You must have the Java SE Development Kit (JDK) 6 Update 26 or higher installed on your system if you want to use the JavaFX SDK.
- The third option is the JavaFX Runtime only. The JavaFX Runtime provides the runtime environment required for running JavaFX desktop applications and applets.
- The next option to download is NetBeans 7.1 with the JDK and JavaFX. This option enables you to start developing JavaFX applications with the NetBeans IDE for building, previewing, and debugging JavaFX applications.
- JavaFX Scene Builder (beta release) is a GUI editor that enables you to develop interfaces in JavaFX FXML.

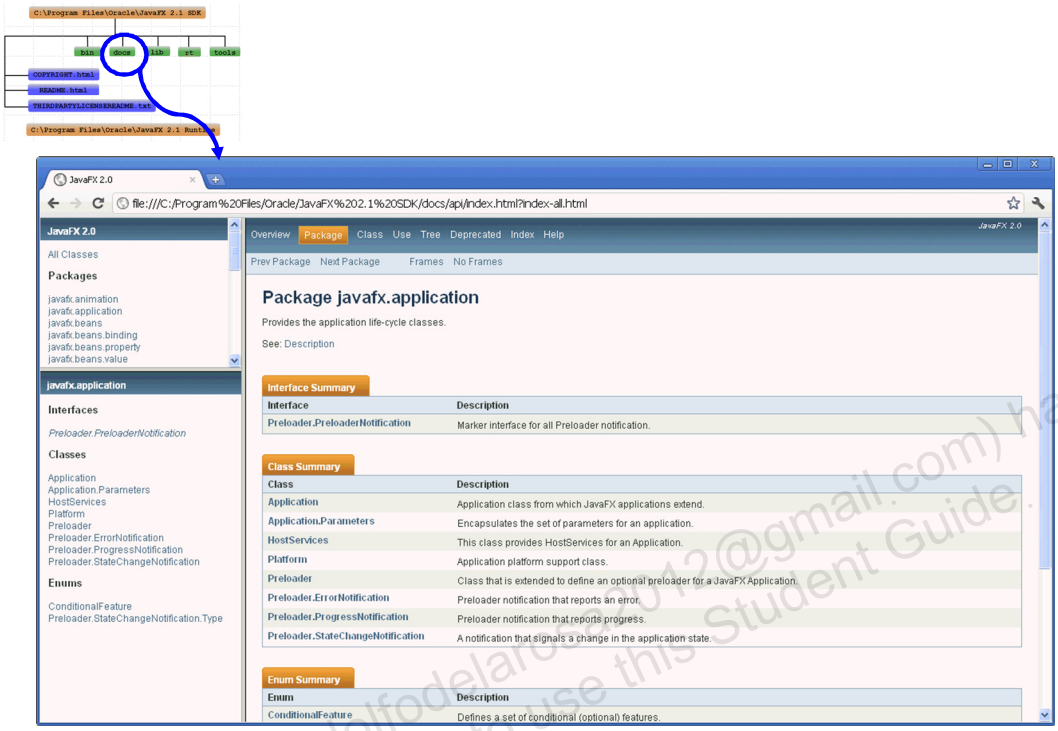
JavaFX Default Installation Directory



- `bin`/SDK build tools
- `docs`/API documentation
- **lib/`Ant`:** Tasks used by NetBeans for packaging and deployment. Developers can use the tools as well. `javafx-doclet.jar: http://docs.oracle.com/javafx/2.0/doclet/jfxpub-doclet.htm ant-javafx.jar: http://docs.oracle.com/javafx/2.0/deployment/deploy_quick_start.htm#JFXDP518`
- **`rt/`:** Contains the `jrxrt.jar` runtime file
- **`tools/Ant`:** Tasks used by NetBeans for packaging and deployment. Developers can use the tools as well. `javafx-doclet.jar: http://docs.oracle.com/javafx/2.0/doclet/jfxpub-doclet.htm ant-javafx.jar: http://docs.oracle.com/javafx/2.0/deployment/deploy_quick_start.htm#JFXDP518 (Note: this directory is maintained for backward compatibility with older versions of NetBeans.)`

- **`COPYRIGHT.html`:** Copyright information for the `readme.html` document
- **`README.html`:** Contains a link to the OTN readme page for Java SE, JavaFX Runtime, and JavaFX SDK
- **`THIRDPARTYLICENSEREADME.txt`:** License information for third-party software included in the JavaFX SDK and the documentation

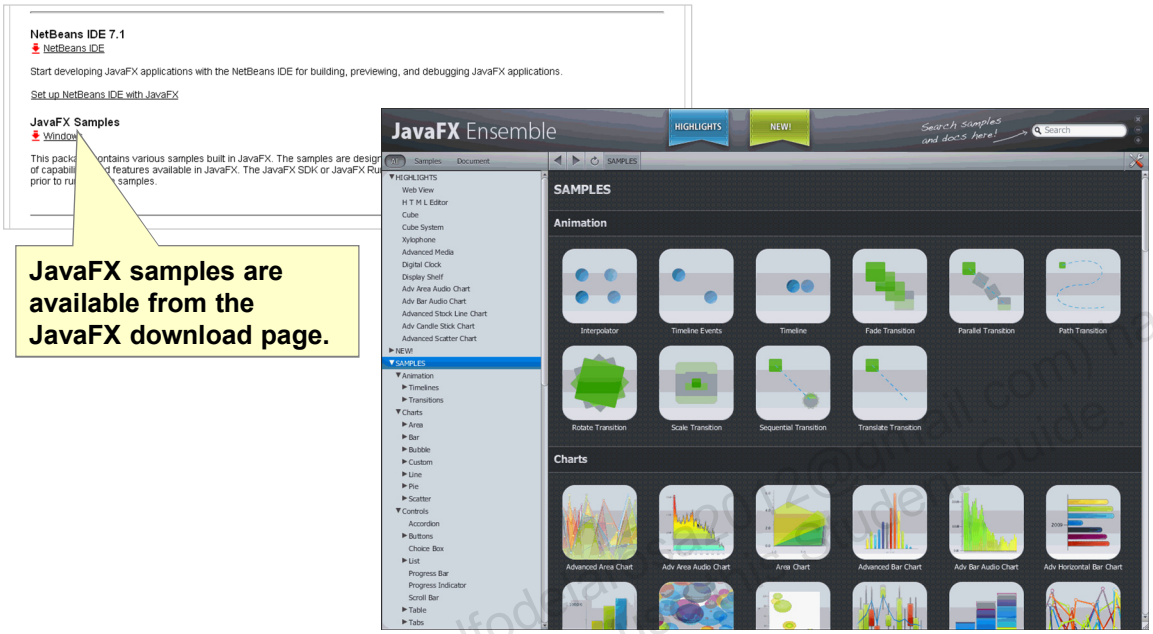
JavaFX API Documentation



The JavaFX API documentation is structured in the same manner as the Java API documentation.

Download JavaFX Samples

Download JavaFX samples from oracle.com/javafx.



JavaFX samples are a good way to get started using JavaFX. The free samples are available on the main JavaFX download page of oracle.com/javafx and are updated with each JavaFX release.

Resources

- oracle.com/javafx
  - JavaFX downloads
  - Samples and technical videos
  - Documentation
  - Blogs and forums
- Oracle Learning Library (oracle.com/oll)
  - Java and JavaFX tutorials, demos, interactive training, and OBEs (Oracle by Example)
- *Java Magazine* (by subscription)

To stay current with JavaFX and its latest releases, you can bookmark oracle.com/javafx, where you can find many JavaFX resources, including:

- Downloadable software
- Samples
- Documentation
- Blogs: http://blogs.oracle.com/javafx/
- Forums: https://forums.oracle.com/forums/forum.jspa?forumID=1385
- Technical videos: http://otn.oracle.com/javafx/

In addition, the Oracle Learning Library at oracle.com/oll has the following kinds of material:

- Tutorials
- Videos and demos
- OBEs (Oracle by Example): Java curriculum developers have developed several OBE (Oracle by Example) tutorials that describe how to use the new Swing enhancements. See http://www.oracle.com/goto/oll and search for “Java” to find tutorials, videos, and OBEs.
- Interactive learning modules

- Links to related Oracle University courses
- Subscription to *Java Magazine*:
  http://www.oracle.com/technetwork/java/javamagazine/index.html

Summary

In this lesson, you should have learned how to:

- Describe the features of JavaFX
- Identify the features of the JavaFX scene graph
- Describe the JavaFX development tools
- Describe how JavaFX is integrated into a Java application
- Use FXML and determine when to use it in an application
- Download and install JavaFX and related samples