# Hands On Java FX

LINK: https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#JFXST784

JavaFX is a set of graphics and media packages that enables developers to design, create, test, debug, and deploy rich client applications that operate consistently across diverse platforms.

## **JavaFX Applications**

Since the JavaFX library is written as a Java API, JavaFX application code can reference APIs from any Java library. For example, JavaFX applications can use Java API libraries to access native system capabilities and connect to server-based middleware applications.

The look and feel of JavaFX applications can be customized. Cascading Style Sheets (CSS) separate appearance and style from implementation so that developers can concentrate on coding. Graphic designers can easily customize the appearance and style of the application through the CSS. *If you have a web design background, or if you would like to separate the user interface (UI) and the back-end logic, then you can develop the presentation aspects of the UI in the FXML scripting language and use Java code for the application logic. If you prefer to design UIs without writing code, then use JavaFX Scene Builder. As you design the UI, Scene Builder creates FXML markup that can be ported to an Integrated Development Environment (IDE) so that developers can add the business logic.*

## **Availability**

The JavaFX APIs are available as a fully integrated feature of the Java SE Runtime Environment (JRE) and the Java Development Kit (JDK ). Because the JDK is available for all major desktop platforms (Windows, Mac OS X, and Linux), JavaFX applications compiled to JDK 7 and later also run on all the major desktop platforms. Support for ARM platforms has also been made available with JavaFX 8. JDK for ARM includes the base, graphics and controls components of JavaFX.

The cross-platform compatibility enables a consistent runtime experience for JavaFX applications developers and users. Oracle ensures synchronized releases and updates on all platforms and offers an [extensive support program](http://www.oracle.com/us/technologies/java/standard-edition/support/overview/) for companies that run mission-critical applications.

On the [JDK download page](http://www.oracle.com/technetwork/java/javase/downloads/), you can get a zip file of JavaFX sample applications. The sample applications provide many code samples and snippets that show by example how to write JavaFX applications. See ["How Do I Run a Sample Application?"](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#BABHGFAH) for more information.

Download Page Links:

* <https://gluonhq.com/products/javafx/>

## **Key Features**

The following features are included in JavaFX 8 and later releases. Items that were introduced in JavaFX 8 release are indicated accordingly:

* **Java APIs**. JavaFX is a Java library that consists of classes and interfaces that are written in Java code. The APIs are designed to be a friendly alternative to Java Virtual Machine (Java VM) languages, such as JRuby and Scala.
* **FXML and Scene Builder**. FXML is an XML-based declarative markup language for constructing a JavaFX application user interface. A designer can code in FXML or use JavaFX Scene Builder to interactively design the graphical user interface (GUI). Scene Builder generates FXML markup that can be ported to an IDE where a developer can add the business logic.
* **WebView**. A web component that uses WebKitHTML technology to make it possible to embed web pages within a JavaFX application. JavaScript running in WebView can call Java APIs, and Java APIs can call JavaScript running in WebView. Support for additional HTML5 features, including Web Sockets, Web Workers, and Web Fonts, and printing capabilities have been added in JavaFX 8. See [Adding HTML Content to JavaFX Applications](https://docs.oracle.com/javase/8/javafx/embedded-browser-tutorial/overview.htm#JFXWV135).
* **Swing interoperability**. Existing Swing applications can be updated with JavaFX features, such as rich graphics media playback and embedded Web content. The SwingNode class, which enables you to embed Swing content into JavaFX applications, has been added in JavaFX 8. See the [SwingNode API javadoc](https://docs.oracle.com/javase/8/javafx/api/) and [Embedding Swing Content in JavaFX Applications](https://docs.oracle.com/javase/8/javafx/interoperability-tutorial/embed-swing.htm#JFXIP566) for more information.
* **Built-in UI controls** **and CSS**. JavaFX provides all the major UI controls that are required to develop a full-featured application. Components can be skinned with standard Web technologies such as CSS. The DatePicker and TreeTableView UI controls are now available with the JavaFX 8 release. See [Using JavaFX UI Controls](https://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui_controls.htm#JFXUI336) for more information. Also, the CSS Styleable\* classes have become public API, allowing objects to be styled by CSS.
* **Modena theme*.*** The Modena theme replaces the Caspian theme as the default for JavaFX 8 applications. The Caspian theme is still available for your use by adding the setUserAgentStylesheet(STYLESHEET\_CASPIAN) line in your Application start() method. For more information, see the [Modena blog](http://fxexperience.com/2013/01/modena-new-theme-for-javafx-8/) at fxexperience.com
* **3D Graphics Features**. The new API classes for Shape3D (Box, Cylinder, MeshView, and Sphere subclasses), SubScene, Material, PickResult, LightBase (AmbientLight and PointLight subclasses), and SceneAntialiasing have been added to the 3D Graphics library in JavaFX 8. The Camera API class has also been updated in this release. For more information, see the [Getting Started with JavaFX 3D Graphics](https://docs.oracle.com/javase/8/javafx/graphics-tutorial/javafx-3d-graphics.htm#JFXGR256) document and the corresponding [API javadoc](https://docs.oracle.com/javase/8/javafx/api/) for javafx.scene.shape.Shape3D, javafx.scene.SubScene, javafx.scene.paint.Material, javafx.scene.input.PickResult, and javafx.scene.SceneAntialiasing.
* **Canvas API**. The Canvas API enables drawing directly within an area of the JavaFX scene that consists of one graphical element (node).
* **Printing API.** The javafx.print package has been added in Java SE 8 release and provides the public classes for the [JavaFX Printing API](https://docs.oracle.com/javase/8/javafx/api/).
* **Rich Text Support**. JavaFX 8 brings enhanced text support to JavaFX, including bi-directional text and complex text scripts, such as Thai and Hindu in controls, and multi-line, multi-style text in text nodes.
* **Multitouch Support**. JavaFX provides support for multitouch operations, based on the capabilities of the underlying platform.
* **Hi-DPI support.** JavaFX 8 now supports Hi-DPI displays.
* **Hardware-accelerated graphics pipeline**. JavaFX graphics are based on the graphics rendering pipeline (Prism). JavaFX offers smooth graphics that render quickly through Prism when it is used with a supported graphics card or graphics processing unit (GPU). If a system does not feature one of the recommended GPUs supported by JavaFX, then Prism defaults to the software rendering stack.
* **High-performance media engine**. The media pipeline supports the playback of web multimedia content. It provides a stable, low-latency media framework that is based on the GStreamer multimedia framework.
* **Self-contained application deployment** **model.** Self-contained application packages have all of the application resources and a private copy of the Java and JavaFX runtimes. They are distributed as native installable packages and provide the same installation and launch experience as native applications for that operating system.

## **How Do I Create a JavaFX Application?**

Because JavaFX applications are written in the Java language, you can use your favorite editor or any integrated development environment (IDE) that supports the Java language (such as NetBeans, Eclipse, or IntelliJ IDEA) to create JavaFX applications.

**To create JavaFX applications:**

1. Go to the Java SE Downloads page at <http://www.oracle.com/technetwork/java/javase/downloads/> to download the Oracle® JDK 8 with JavaFX 8.*n*support. Links to the certified system configurations and release notes are also available on that page..
2. Use [Getting Started with JavaFX Sample Applications](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/get_start_apps.htm#BACECIIB) to create simple applications that demonstrates how to work with layouts, style sheets, and visual effects.
3. Use JavaFX Scene Builder to design the UI for your JavaFX application without coding. You can drag and drop UI components to a work area, modify their properties, apply style sheets, and integrate the resulting code with their application logic.
   1. Download the JavaFX Scene Builder from the JavaFX Downloads page at <http://www.oracle.com/technetwork/java/javase/downloads/>.
   2. Follow the Getting Started with JavaFX Scene Builder tutorial to learn more.

# **Getting Started with JavaFX Sample Applications**

Here are the important things to know about the basic structure of a JavaFX application:

* The main class for a JavaFX application extends the javafx.application.Application class. The start() method is the main entry point for all JavaFX applications.
* A JavaFX application defines the user interface container by means of a stage and a scene. The JavaFX Stage class is the top-level JavaFX container. The JavaFX Scene class is the container for all content. [Example 3-1](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/hello_world.htm#CHDIFJHE) creates the stage and scene and makes the scene visible in a given pixel size.
* In JavaFX, the content of the scene is represented as a hierarchical scene graph of nodes. In this example, the root node is a StackPane object, which is a resizable layout node. This means that the root node's size tracks the scene's size and changes when the stage is resized by a user.
* The root node contains one child node, a button control with text, plus an event handler to print a message when the button is pressed.
* The main() method is not required for JavaFX applications when the JAR file for the application is created with the JavaFX Packager tool, which embeds the JavaFX Launcher in the JAR file. However, it is useful to include the main() method so you can run JAR files that were created without the JavaFX Launcher, such as when using an IDE in which the JavaFX tools are not fully integrated. Also, Swing applications that embed JavaFX code require the main() method.

# **Creating a Form in JavaFX**

Creating a form is a common activity when developing an application. This tutorial teaches you the basics of screen layout, how to add controls to a layout pane, and how to create input events.

In this tutorial, you will use JavaFX to build the login form shown in [Figure 4-1](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/form.htm#BABJHICA).

***Figure 4-1 Login Form***

  
[Description of "Figure 4-1 Login Form"](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/img_text/login.htm)

The tool used in this Getting Started tutorial is NetBeans IDE. Before you begin, ensure that the version of NetBeans IDE that you are using supports JavaFX 8. See the Certified System Configurations page of the [Java SE Downloads page](http://www.oracle.com/technetwork/java/javase/downloads/) for details.

## **Create the Project**

Your first task is to create a JavaFX project in NetBeans IDE and name it Login:

1. From the **File** menu, choose **New Project**.
2. In the **JavaFX** application category, choose **JavaFX Application**. Click **Next**.
3. Name the project **Login** and click **Finish**.

When you create a JavaFX project, NetBeans IDE provides a Hello World application as a starting point, which you have already seen if you followed the [Hello World](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/hello_world.htm) tutorial.

1. Remove the start() method that NetBeans IDE generated and replace it with the code in [Example 4-1](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/form.htm#CFHIBBBH).

***Example 4-1 Application Stage***

@Override

public void start(Stage primaryStage) {

primaryStage.setTitle("JavaFX Welcome");

primaryStage.show();

}

**Tip:** After you add sample code into a NetBeans project, press Ctrl (or Cmd) + Shift + I to import the required packages. When there is a choice of import statements, choose the one that starts with javafx.

## **Create a GridPane Layout**

For the login form, use a GridPane layout because it enables you to create a flexible grid of rows and columns in which to lay out controls. You can place controls in any cell in the grid, and you can make controls span cells as needed.

The code to create the GridPane layout is in [Example 4-2](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/form.htm#CFHEAHGB). Add the code before the line primaryStage.show();

***Example 4-2 GridPane with Gap and Padding Properties***

GridPane grid = new GridPane();

grid.setAlignment(Pos.CENTER);

grid.setHgap(10);

grid.setVgap(10);

grid.setPadding(new Insets(25, 25, 25, 25));

Scene scene = new Scene(grid, 300, 275);

primaryStage.setScene(scene);

[Example 4-2](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/form.htm#CFHEAHGB) creates a GridPane object and assigns it to the variable named grid. The alignment property changes the default position of the grid from the top left of the scene to the center. The gap properties manage the spacing between the rows and columns, while the padding property manages the space around the edges of the grid pane. The insets are in the order of top, right, bottom, and left. In this example, there are 25 pixels of padding on each side.

The scene is created with the grid pane as the root node, which is a common practice when working with layout containers. Thus, as the window is resized, the nodes within the grid pane are resized according to their layout constraints. In this example, the grid pane remains in the center when you grow or shrink the window. The padding properties ensure there is a padding around the grid pane when you make the window smaller.

This code sets the scene width and height to 300 by 275. If you do not set the scene dimensions, the scene defaults to the minimum size needed to display its contents.

## **Add Text, Labels, and Text Fields**

Looking at [Figure 4-1](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/form.htm#BABJHICA), you can see that the form requires the title ”Welcome ”and text and password fields for gathering information from the user. The code for creating these controls is in [Example 4-3](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/form.htm#CFHFDFHJ). Add this code after the line that sets the grid padding property.

***Example 4-3 Controls***

Text scenetitle = new Text("Welcome");

scenetitle.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));

grid.add(scenetitle, 0, 0, 2, 1);

Label userName = new Label("User Name:");

grid.add(userName, 0, 1);

TextField userTextField = new TextField();

grid.add(userTextField, 1, 1);

Label pw = new Label("Password:");

grid.add(pw, 0, 2);

PasswordField pwBox = new PasswordField();

grid.add(pwBox, 1, 2);

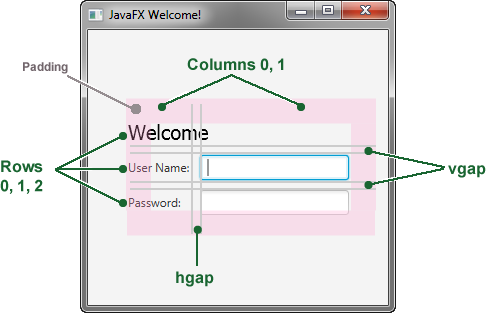
The first line creates a Text object that cannot be edited, sets the text to Welcome, and assigns it to a variable named scenetitle. The next line uses the setFont()method to set the font family, weight, and size of the scenetitle variable. Using an inline style is appropriate where the style is bound to a variable, but a better technique for styling the elements of your user interface is by using a style sheet. In the next tutorial, [Fancy Forms with JavaFX CSS](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/css.htm#BABBGJBI), you will replace the inline style with a style sheet.

The grid.add() method adds the scenetitle variable to the layout grid. The numbering for columns and rows in the grid starts at zero, and scenetitle is added in column 0, row 0. The last two arguments of the grid.add() method set the column span to 2 and the row span to 1.

The next lines create a Label object with text User Name at column 0, row 1 and a Text Field object that can be edited. The text field is added to the grid pane at column 1, row 1. A password field and label are created and added to the grid pane in a similar fashion.

When working with a grid pane, you can display the grid lines, which is useful for debugging purposes. In this case, you can add grid.setGridLinesVisible(true)after the line that adds the password field. Then, when you run the application, you see the lines for the grid columns and rows as well as the gap properties, as shown in [Figure 4-2](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/form.htm#BABFFDCJ).

***Figure 4-2 Login Form with Grid Lines***

  
[Description of "Figure 4-2 Login Form with Grid Lines"](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/img_text/login_gridlines_co.htm)

## **Add a Button and Text**

The final two controls required for the application are a Button control for submitting the data and a Text control for displaying a message when the user presses the button.

First, create the button and position it on the bottom right, which is a common placement for buttons that perform an action affecting the entire form. The code is in [Example 4-4](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/form.htm#CFHIJGHC). Add this code before the code for the scene.

***Example 4-4 Button***

Button btn = new Button("Sign in");

HBox hbBtn = new HBox(10);

hbBtn.setAlignment(Pos.BOTTOM\_RIGHT);

hbBtn.getChildren().add(btn);

grid.add(hbBtn, 1, 4);

The first line creates a button named btn with the label Sign in, and the second line creates an HBox layout pane named hbBtn with spacing of 10 pixels. The HBoxpane sets an alignment for the button that is different from the alignment applied to the other controls in the grid pane. The alignment property has a value of Pos.BOTTOM\_RIGHT, which positions a node at the bottom of the space vertically and at the right edge of the space horizontally. The button is added as a child of the HBox pane, and the HBox pane is added to the grid in column 1, row 4.

Now, add a Text control for displaying the message, as shown in [Example 4-5](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/form.htm#BABHGHFI). Add this code before the code for the scene.

***Example 4-5 Text***

final Text actiontarget = new Text();

grid.add(actiontarget, 1, 6);

[Figure 4-3](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/form.htm#BABFICHF) shows the form now. You will not see the text message until you work through the next section of the tutorial, [Add Code to Handle an Event](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/form.htm#BABHABBF).

***Figure 4-3 Login Form with Button***

  
[Description of "Figure 4-3 Login Form with Button"](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/img_text/login.htm)

## **Add Code to Handle an Event**

Finally, make the button display the text message when the user presses it. Add the code in [Example 4-6](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/form.htm#CFHDCFFD) before the code for the scene.

***Example 4-6 Button Event***

btn.setOnAction(new EventHandler<ActionEvent>() {

@Override

public void handle(ActionEvent e) {

actiontarget.setFill(Color.FIREBRICK);

actiontarget.setText("Sign in button pressed");

}

});

The setOnAction() method is used to register an event handler that sets the actiontarget object to Sign in button pressed when the user presses the button. The color of the actiontarget object is set to firebrick red.

## **Run the Application**

Right-click the **Login** project node in the Projects window, choose **Run**, and then click the Sign in button. [Figure 4-4](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/form.htm#BABIAIFI) shows the results. If you run into problems, then take a look at the code in the Login.java file that is included in the downloadable [Login.zip](https://docs.oracle.com/javase/8/javafx/sample-apps/Login.zip) file.

***Figure 4-4 Final Login Form***



# **5 Fancy Forms with JavaFX CSS**

This tutorial is about making your JavaFX application look attractive by adding a Cascading Style Sheet (CSS). You develop a design, create a .css file, and apply the new styles.

In this tutorial, you will take a Login form that uses default styles for labels, buttons, and background color, and, with a few simple CSS modifications, turn it into a stylized application, as shown in [Figure 5-1](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/css.htm#BEIHJCFB).

***Figure 5-1 Login Form With and Without CSS***



## **Create the CSS File**

Your first task is to create a new CSS file and save it in the same directory as the main class of your application. After that, you must make the JavaFX application aware of the newly added Cascading Style Sheet.

1. In the NetBeans IDE Projects window, expand the **Login** project node and then the **Source Packages** directory node.
2. Right-click the **login** folder under the Source Packages directory and choose **New**, then **Other**.
3. In the New File dialog box, choose **Other**, then **Cascading Style Sheet**, and click **Next**.
4. Enter **Login** for the File Name text field and ensure the Folder text field value is src\login.
5. Click **Finish**.
6. In the Login.java file, initialize the style sheets variable of the Scene class to point to the Cascading Style Sheet by including the line of code shown in bold below so that it appears as shown in [Example 5-1](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/css.htm#BABBJFAH).

***Example 5-1 Initialize the stylesheets Variable***

Scene scene = new Scene(grid, 300, 275);

primaryStage.setScene(scene);

**scene.getStylesheets().add**

**(Login.class.getResource("Login.css").toExternalForm());**

primaryStage.show();

This code looks for the style sheet in the src\login directory in the NetBeans project.

For More Information About Loading the css file from external resources see below link

* <https://blog.idrsolutions.com/2014/04/use-external-css-files-javafx/>
* http://www.java2s.com/Code/Java/JavaFX/ConnecttoCSSStyleSheetinanotherPackage.htm

## **Add a Background Image**

A background image helps make your form more attractive. For this tutorial, you add a gray background with a linen-like texture.

First, download the background image by right-clicking the [background.jpg](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/backgroundjpg.htm#CEGBAIFI) image and saving it into the src\login folder in the Login NetBeans project.

Now, add the code for the background-image property to the CSS file. Remember that the path is relative to the style sheet. So, in the code in [Example 5-2](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/css.htm#BABDHCFE), the background.jpg image is in the same directory as the Login.css file.

***Example 5-2 Background Image***

.root {

-fx-background-image: url("background.jpg");

}

## **Style the Labels**

The next controls to enhance are the labels. You will use the .label style class, which means the styles will affect all labels in the form. The code is in [Example 5-3](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/css.htm#BEIIFIEG).

***Example 5-3 Font Size, Fill, Weight, and Effect on Labels***

.label {

-fx-font-size: 12px;

-fx-font-weight: bold;

-fx-text-fill: #333333;

-fx-effect: dropshadow( gaussian , rgba(255,255,255,0.5) , 0,0,0,1 );

}

This example increases the font size and weight and applies a drop shadow of a gray color (#333333). The purpose of the drop shadow is to add contrast between the dark gray text and the light gray background. See the section on effects in the [JavaFX CSS Reference Guide](https://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html#typeeffect) for details on the parameters of the drop shadow property.

## **Style Text**

Now, create some special effects on the two Text objects in the form: scenetitle, which includes the text Welcome, and actiontarget, which is the text that is returned when the user presses the Sign in button. You can apply different styles to Text objects used in such diverse ways.

1. In the Login.java file, remove the following lines of code that define the inline styles currently set for the text objects:

scenetitle.setFont(Font.font(”Tahoma”, FontWeight.NORMAL, 20));

actiontarget.setFill(Color.FIREBRICK);

By switching to CSS over inline styles, you separate the design from the content. This approach makes it easier for a designer to have control over the style without having to modify content.

1. Create an ID for each text node by using the setID() method of the Node class: Add the following lines in bold so that they appear as shown in [Example 5-4](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/css.htm#CHDHGGEI).
2. ***Example 5-4 Create ID for Text Nodes***
3. ...
4. Text scenetitle = new Text("Welcome");
5. **scenetitle.setId("welcome-text");**
6. ...
7. ...
8. grid.add(actiontarget, 1, 6);
9. **actiontarget.setId("actiontarget");**
10. ..
11. In the Login.css file, define the style properties for the welcome-text and actiontarget IDs. For the style name, use the ID preceded by a number sign (#), as shown in [Example 5-5](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/css.htm#BABEHDBB).

***Example 5-5 Text Effect***

#welcome-text {

-fx-font-size: 32px;

-fx-font-family: "Arial Black";

-fx-fill: #818181;

-fx-effect: innershadow( three-pass-box , rgba(0,0,0,0.7) , 6, 0.0 , 0 , 2 );

}

#actiontarget {

-fx-fill: FIREBRICK;

-fx-font-weight: bold;

-fx-effect: dropshadow( gaussian , rgba(255,255,255,0.5) , 0,0,0,1 );

}

The size of the Welcome text is increased to 32 points and the font is changed to Arial Black. The text fill color is set to a dark gray color (#818181) and an inner shadow effect is applied, creating an embossing effect. You can apply an inner shadow to any color by changing the text fill color to be a darker version of the background. See the section on effects in the [JavaFX CSS Reference Guide](https://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html#typeeffect) for details about the parameters of inner shadow property.

The style definition for actiontarget is similar to what you have seen before.

## **Style the Button**

The next step is to style the button, making it change style when the user hovers the mouse over it. This change will give users an indication that the button is interactive, a standard design practice.

First, create the style for the initial state of the button by adding the code in [Example 5-6](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/css.htm#BABCCDDI). This code uses the **.button style class selector,** such that if you add a button to the form at a later date, then the new button will also use this style.

Now, create a slightly different look for when the user hovers the mouse over the button. You do this with the hover pseudo-class. A pseudo-class includes the selector for the class and the name for the state separated by a colon (:), as shown in [Example 5-7](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/css.htm#BEIHDDGE).

***Example 5-7 Button Hover Style***

.button:hover {

-fx-background-color: linear-gradient(#2A5058, #61a2b1);

}

[Figure 5-5](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/css.htm#BEIFGBEB) shows the initial and hover states of the button with its new blue-gray background and white bold text.

***Figure 5-5 Initial and Hover Button States***

  
[Description of "Figure 5-5 Initial and Hover Button States"](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/img_text/buttons2.htm)

## **Where to Go from Here**

Here are some things for you to try next:

* See what you can create using CSS. Some documents that can help you are [Skinning JavaFX Applications with CSS](https://docs.oracle.com/javase/8/javafx/user-interface-tutorial/css_tutorial.htm#JFXUI733), [Styling Charts with CSS](https://docs.oracle.com/javase/8/javafx/user-interface-tutorial/css-styles.htm#JFXUI672), and the [JavaFX CSS Reference Guide.](https://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html) The Skinning with CSS and the CSS Analyzer section of the JavaFX Scene Builder User Guide also provides information on how you can use the JavaFX Scene Builder tool to skin your JavaFX FXML layout.
* See [Styling FX Buttons with CSS](http://fxexperience.com/2011/12/styling-fx-buttons-with-css/) for examples of how to create common button styles using CSS.

# **Styling FX Buttons with CSS**

#windows7-default {

    -fx-background-color:

        #3c7fb1,

        linear-gradient(#fafdfe, #e8f5fc),

        linear-gradient(#eaf6fd 0%, #d9f0fc 49%, #bee6fd 50%, #a7d9f5 100%);

    -fx-background-insets: 0,1,2;

    -fx-background-radius: 3,2,1;

    -fx-padding: 3 30 3 30;

    -fx-text-fill: black;

    -fx-font-size: 14px;

}

The first line defines the 3 background fills, first is solid color and the other two are linear gradients. The background-insets offsets the backgrounds so they do not 100% paint over each other and the second background is 1px in from the outside and the 3rd background is 2px in from the outside of the button. The background-radius us setting the corner radius’s of the 3 backgrounds getting smaller as the backgrounds move in, this makes the gap between the borders a consistent 1 pixel all the way around. Padding adds extra space around the text to make the button bigger by default. Then the last two lines set the text color and size. That is all there is too it.

I have not included styles for the pressed, over and focused states of all the buttons but they can all be easily added in a similar way. I have included a over and pressed state for the “Record Sales” button as a example.

Here is the CSS code of all the button styles you see above:

#green {

    -fx-background-color:

        linear-gradient(#f0ff35, #a9ff00),

        radial-gradient(center 50% -40%, radius 200%, #b8ee36 45%, #80c800 50%);

    -fx-background-radius: 6, 5;

    -fx-background-insets: 0, 1;

    -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.4) , 5, 0.0 , 0 , 1 );

    -fx-text-fill: #395306;

}

#round-red {

    -fx-background-color: linear-gradient(#ff5400, #be1d00);

    -fx-background-radius: 30;

    -fx-background-insets: 0;

    -fx-text-fill: white;

}

#bevel-grey {

    -fx-background-color:

        linear-gradient(#f2f2f2, #d6d6d6),

        linear-gradient(#fcfcfc 0%, #d9d9d9 20%, #d6d6d6 100%),

        linear-gradient(#dddddd 0%, #f6f6f6 50%);

    -fx-background-radius: 8,7,6;

    -fx-background-insets: 0,1,2;

    -fx-text-fill: black;

    -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.6) , 5, 0.0 , 0 , 1 );

}

#glass-grey {

    -fx-background-color:

        #c3c4c4,

        linear-gradient(#d6d6d6 50%, white 100%),

        radial-gradient(center 50% -40%, radius 200%, #e6e6e6 45%, rgba(230,230,230,0) 50%);

    -fx-background-radius: 30;

    -fx-background-insets: 0,1,1;

    -fx-text-fill: black;

    -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.6) , 3, 0.0 , 0 , 1 );

}

#shiny-orange {

    -fx-background-color:

        linear-gradient(#ffd65b, #e68400),

        linear-gradient(#ffef84, #f2ba44),

        linear-gradient(#ffea6a, #efaa22),

        linear-gradient(#ffe657 0%, #f8c202 50%, #eea10b 100%),

        linear-gradient(from 0% 0% to 15% 50%, rgba(255,255,255,0.9), rgba(255,255,255,0));

    -fx-background-radius: 30;

    -fx-background-insets: 0,1,2,3,0;

    -fx-text-fill: #654b00;

    -fx-font-weight: bold;

    -fx-font-size: 14px;

    -fx-padding: 10 20 10 20;

}

#dark-blue {

    -fx-background-color:

        #090a0c,

        linear-gradient(#38424b 0%, #1f2429 20%, #191d22 100%),

        linear-gradient(#20262b, #191d22),

        radial-gradient(center 50% 0%, radius 100%, rgba(114,131,148,0.9), rgba(255,255,255,0));

    -fx-background-radius: 5,4,3,5;

    -fx-background-insets: 0,1,2,0;

    -fx-text-fill: white;

    -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.6) , 5, 0.0 , 0 , 1 );

    -fx-font-family: "Arial";

    -fx-text-fill: linear-gradient(white, #d0d0d0);

    -fx-font-size: 12px;

    -fx-padding: 10 20 10 20;

}

#dark-blue Text {

    -fx-effect: dropshadow( one-pass-box , rgba(0,0,0,0.9) , 1, 0.0 , 0 , 1 );

}

#record-sales {

    -fx-padding: 8 15 15 15;

    -fx-background-insets: 0,0 0 5 0, 0 0 6 0, 0 0 7 0;

    -fx-background-radius: 8;

    -fx-background-color:

        linear-gradient(from 0% 93% to 0% 100%, #a34313 0%, #903b12 100%),

        #9d4024,

        #d86e3a,

        radial-gradient(center 50% 50%, radius 100%, #d86e3a, #c54e2c);

    -fx-effect: dropshadow( gaussian , rgba(0,0,0,0.75) , 4,0,0,1 );

    -fx-font-weight: bold;

    -fx-font-size: 1.1em;

}

#record-sales:hover {

    -fx-background-color:

        linear-gradient(from 0% 93% to 0% 100%, #a34313 0%, #903b12 100%),

        #9d4024,

        #d86e3a,

        radial-gradient(center 50% 50%, radius 100%, #ea7f4b, #c54e2c);

}

#record-sales:pressed {

    -fx-padding: 10 15 13 15;

    -fx-background-insets: 2 0 0 0,2 0 3 0, 2 0 4 0, 2 0 5 0;

}

#record-sales Text {

    -fx-fill: white;

    -fx-effect: dropshadow( gaussian , #a30000 , 0,0,0,2 );

}

#rich-blue {

    -fx-background-color:

        #000000,

        linear-gradient(#7ebcea, #2f4b8f),

        linear-gradient(#426ab7, #263e75),

        linear-gradient(#395cab, #223768);

    -fx-background-insets: 0,1,2,3;

    -fx-background-radius: 3,2,2,2;

    -fx-padding: 12 30 12 30;

    -fx-text-fill: white;

    -fx-font-size: 12px;

}

#rich-blue Text {

    -fx-effect: dropshadow( one-pass-box , rgba(0,0,0,0.8) , 0, 0.0 , 0 , 1);

}

#big-yellow {

    -fx-background-color:

        #ecebe9,

        rgba(0,0,0,0.05),

        linear-gradient(#dcca8a, #c7a740),

        linear-gradient(#f9f2d6 0%, #f4e5bc 20%, #e6c75d 80%, #e2c045 100%),

        linear-gradient(#f6ebbe, #e6c34d);

    -fx-background-insets: 0,9 9 8 9,9,10,11;

    -fx-background-radius: 50;

    -fx-padding: 15 30 15 30;

    -fx-font-family: "Helvetica";

    -fx-font-size: 18px;

    -fx-text-fill: #311c09;

    -fx-effect: innershadow( three-pass-box , rgba(0,0,0,0.1) , 2, 0.0 , 0 , 1);

}

#big-yellow Text {

    -fx-effect: dropshadow( one-pass-box , rgba(255,255,255,0.5) , 0, 0.0 , 0 , 1);

}

#iphone-toolbar {

    -fx-background-color: linear-gradient(#98a8bd 0%, #8195af 25%, #6d86a4 100%);

}

#iphone {

    -fx-background-color:

        #a6b5c9,

        linear-gradient(#303842 0%, #3e5577 20%, #375074 100%),

        linear-gradient(#768aa5 0%, #849cbb 5%, #5877a2 50%, #486a9a 51%, #4a6c9b 100%);

    -fx-background-insets: 0 0 -1 0,0,1;

    -fx-background-radius: 5,5,4;

    -fx-padding: 7 30 7 30;

    -fx-text-fill: #242d35;

    -fx-font-family: "Helvetica";

    -fx-font-size: 12px;

    -fx-text-fill: white;

}

#iphone Text {

    -fx-effect: dropshadow( one-pass-box , rgba(0,0,0,0.8) , 0, 0.0 , 0 , -1 );

}

#ipad-dark-grey {

    -fx-background-color:

        linear-gradient(#686868 0%, #232723 25%, #373837 75%, #757575 100%),

        linear-gradient(#020b02, #3a3a3a),

        linear-gradient(#9d9e9d 0%, #6b6a6b 20%, #343534 80%, #242424 100%),

        linear-gradient(#8a8a8a 0%, #6b6a6b 20%, #343534 80%, #262626 100%),

        linear-gradient(#777777 0%, #606060 50%, #505250 51%, #2a2b2a 100%);

    -fx-background-insets: 0,1,4,5,6;

    -fx-background-radius: 9,8,5,4,3;

    -fx-padding: 15 30 15 30;

    -fx-font-family: "Helvetica";

    -fx-font-size: 18px;

    -fx-font-weight: bold;

    -fx-text-fill: white;

    -fx-effect: dropshadow( three-pass-box , rgba(255,255,255,0.2) , 1, 0.0 , 0 , 1);

}

#ipad-dark-grey Text {

    -fx-effect: dropshadow( one-pass-box , black , 0, 0.0 , 0 , -1 );

}

#ipad-grey {

    -fx-background-color:

        linear-gradient(#686868 0%, #232723 25%, #373837 75%, #757575 100%),

        linear-gradient(#020b02, #3a3a3a),

        linear-gradient(#b9b9b9 0%, #c2c2c2 20%, #afafaf 80%, #c8c8c8 100%),

        linear-gradient(#f5f5f5 0%, #dbdbdb 50%, #cacaca 51%, #d7d7d7 100%);

    -fx-background-insets: 0,1,4,5;

    -fx-background-radius: 9,8,5,4;

    -fx-padding: 15 30 15 30;

    -fx-font-family: "Helvetica";

    -fx-font-size: 18px;

    -fx-font-weight: bold;

    -fx-text-fill: #333333;

    -fx-effect: dropshadow( three-pass-box , rgba(255,255,255,0.2) , 1, 0.0 , 0 , 1);

}

#ipad-grey Text {

    -fx-effect: dropshadow( one-pass-box , white , 0, 0.0 , 0 , 1 );

}

#lion-default {

    -fx-background-color:

        rgba(0,0,0,0.08),

        linear-gradient(#5a61af, #51536d),

        linear-gradient(#e4fbff 0%,#cee6fb 10%, #a5d3fb 50%, #88c6fb 51%, #d5faff 100%);

    -fx-background-insets: 0 0 -1 0,0,1;

    -fx-background-radius: 5,5,4;

    -fx-padding: 3 30 3 30;

    -fx-text-fill: #242d35;

    -fx-font-size: 14px;

}

#lion {

    -fx-background-color:

        rgba(0,0,0,0.08),

        linear-gradient(#9a9a9a, #909090),

        linear-gradient(white 0%, #f3f3f3 50%, #ececec 51%, #f2f2f2 100%);

    -fx-background-insets: 0 0 -1 0,0,1;

    -fx-background-radius: 5,5,4;

    -fx-padding: 3 30 3 30;

    -fx-text-fill: #242d35;

    -fx-font-size: 14px;

}

#windows7-default {

    -fx-background-color:

        #3c7fb1,

        linear-gradient(#fafdfe, #e8f5fc),

        linear-gradient(#eaf6fd 0%, #d9f0fc 49%, #bee6fd 50%, #a7d9f5 100%);

    -fx-background-insets: 0,1,2;

    -fx-background-radius: 3,2,1;

    -fx-padding: 3 30 3 30;

    -fx-text-fill: black;

    -fx-font-size: 14px;

}

#windows7 {

    -fx-background-color:

        #707070,

        linear-gradient(#fcfcfc, #f3f3f3),

        linear-gradient(#f2f2f2 0%, #ebebeb 49%, #dddddd 50%, #cfcfcf 100%);

    -fx-background-insets: 0,1,2;

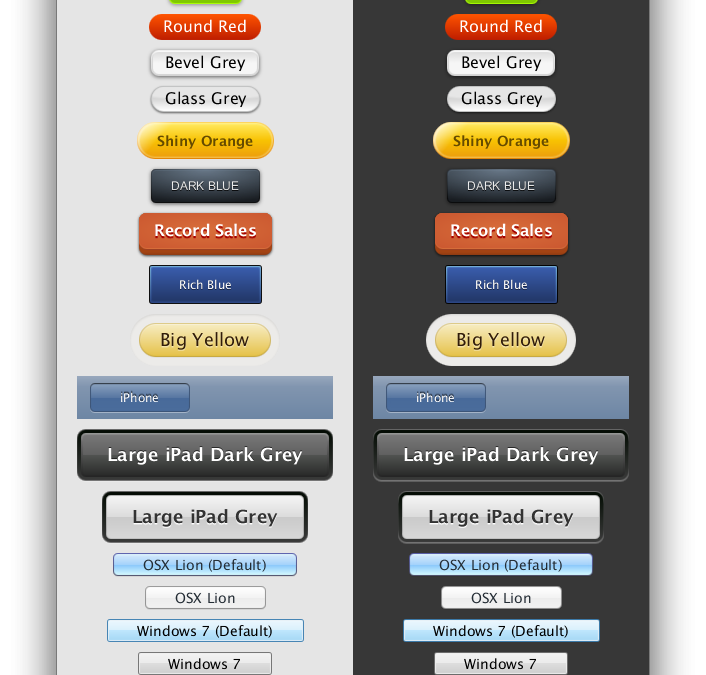
    -fx-background-radius: 3,2,1;

    -fx-padding: 3 30 3 30;

    -fx-text-fill: black;

    -fx-font-size: 14px;

}



# **Animation and Visual Effects in JavaFX**

You can use JavaFX to quickly develop applications with rich user experiences. In this Getting Started tutorial, you will learn to create animated objects and attain complex effects with very little coding.

[Figure 7-1](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/animation.htm#BABFCHDH) shows the application to be created.

***Figure 7-1 Colorful Circles Application***

  
[Description of "Figure 7-1 Colorful Circles Application"](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/img_text/colorfulcircles.htm)

The tool used in this Getting Started tutorial is NetBeans IDE. Before you begin, ensure that the version of NetBeans IDE that you are using supports JavaFX 8. See the Certified System Configurations section of the [Java SE Downloads page](http://www.oracle.com/technetwork/java/javase/downloads/) for details.

## **Set Up the Application**

Set up your JavaFX project in NetBeans IDE as follows:

1. From the **File** menu, choose **New Project**.
2. In the **JavaFX** application category, choose **JavaFX Application**. Click **Next**.
3. Name the project **ColorfulCircles** and click **Finish**.
4. Delete the import statements that NetBeans IDE generated.

You can generate the import statements as you work your way through the tutorial by using either the code completion feature or the Fix Imports command from the Source menu in NetBeans IDE. When there is a choice of import statements, choose the one that starts with javafx.

For the ColorfulCircles application, it is appropriate to use a group node as the root node for the scene. The size of the group is dictated by the size of the nodes within it. For most applications, however, you want the nodes to track the size of the scene and change when the stage is resized. In that case, you use a resizable layout node as the root, as described in [Creating a Form in JavaFX](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/form.htm#BABCIIBJ).

You can compile and run the ColorfulCircles application now, and at each step of the tutorial, to see the intermediate results. If you run into problems, then take a look at the code in the ColorfulCircles.java file, which is included in the downloadable [ColorfulCircles.zip](https://docs.oracle.com/javase/8/javafx/sample-apps/ColorfulCircles.zip) file. At this point, the application is a simple black window.

## **Add Graphics**

Next, create 30 circles by adding the code in [Example 7-2](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/animation.htm#BABEHJGA) before the primaryStage.show() line.

***Example 7-2 30 Circles***

Group circles = new Group();

for (int i = 0; i < 30; i++) {

Circle circle = new Circle(150, Color.web("white", 0.05));

circle.setStrokeType(StrokeType.OUTSIDE);

circle.setStroke(Color.web("white", 0.16));

circle.setStrokeWidth(4);

circles.getChildren().add(circle);

}

root.getChildren().add(circles);

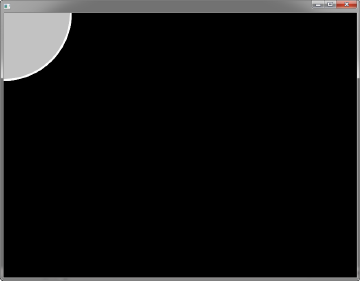
his code creates a group named circles, then uses a for loop to add 30 circles to the group. Each circle has a radius of 150, fill color of white, and opacity level of 5 percent, meaning it is mostly transparent.

To create a border around the circles, the code includes the StrokeType class. A stroke type of OUTSIDE means the boundary of the circle is extended outside the interior by the StrokeWidth value, which is 4. The color of the stroke is white, and the opacity level is 16 percent, making it brighter than the color of the circles.

The final line adds the circles group to the root node. This is a temporary structure. Later, you will modify this scene graph to match the one shown in [Figure 7-2](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/animation.htm#BABBHDBC).

[Figure 7-3](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/animation.htm#BABICCEJ) shows the application. Because the code does not yet specify a unique location for each circle, the circles are drawn on top of one another, with the upper left-hand corner of the window as the center point for the circles. The opacity of the overlaid circles interacts with the black background, producing the gray color of the circles.

***Figure 7-3 Circles***



## **Add a Visual Effect**

Continue by applying a box blur effect to the circles so that they appear slightly out of focus. The code is in [Example 7-3](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/animation.htm#BABEEFEA). Add this code before the primaryStage.show() line.

***Example 7-3 Box Blur Effect***

circles.setEffect(new BoxBlur(10, 10, 3));

## **Create a Background Gradient**

Now, create a rectangle and fill it with a linear gradient, as shown in [Example 7-4](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/animation.htm#BABIICJF).

Add the code before the root.getChildren().add(circles) line so that the gradient rectangle appears behind the circles.

***Example 7-4 Linear Gradient***

Rectangle colors = new Rectangle(scene.getWidth(), scene.getHeight(),

new LinearGradient(0f, 1f, 1f, 0f, true, CycleMethod.NO\_CYCLE, new

Stop[]{

new Stop(0, Color.web("#f8bd55")),

new Stop(0.14, Color.web("#c0fe56")),

new Stop(0.28, Color.web("#5dfbc1")),

new Stop(0.43, Color.web("#64c2f8")),

new Stop(0.57, Color.web("#be4af7")),

new Stop(0.71, Color.web("#ed5fc2")),

new Stop(0.85, Color.web("#ef504c")),

new Stop(1, Color.web("#f2660f")),}));

colors.widthProperty().bind(scene.widthProperty());

colors.heightProperty().bind(scene.heightProperty());

root.getChildren().add(colors);

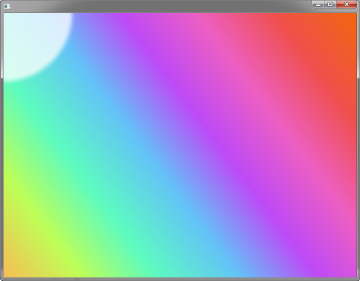
This code creates a rectangle named colors. The rectangle is the same width and height as the scene and is filled with a linear gradient that starts in the lower left-hand corner (0, 1) and ends in the upper right-hand corner (1, 0). The value of true means the gradient is proportional to the rectangle, and NO\_CYCLE indicates that the color cycle will not repeat. The Stop[] sequence indicates what the gradient color should be at a particular spot.

The next two lines of code make the linear gradient adjust as the size of the scene changes by binding the width and height of the rectangle to the width and height of the scene. See [Using JavaFX Properties and Bindings](http://www.oracle.com/pls/topic/lookup?ctx=javase80&id=JFXBD) for more information on binding.

The final line of code adds the colors rectangle to the root node.

The gray circles with the blurry edges now appear on top of a rainbow of colors, as shown in [Figure 7-5](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/animation.htm#BABIECDC).

***Figure 7-5 Linear Gradient***



## **Apply a Blend Mode**

Next, add color to the circles and darken the scene by adding an overlay blend effect. You will remove the circles group and the linear gradient rectangle from the scene graph and add them to the new overlay blend group.

1. Locate the following two lines of code:
2. root.getChildren().add(colors);
3. root.getChildren().add(circles);
4. Replace the two lines of code from Step [1](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/animation.htm#CACECAEG) with the code in [Example 7-5](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/animation.htm#BABEDGDF).

***Example 7-5 Blend Mode***

Group blendModeGroup =

new Group(new Group(new Rectangle(scene.getWidth(), scene.getHeight(),

Color.BLACK), circles), colors);

colors.setBlendMode(BlendMode.OVERLAY);

root.getChildren().add(blendModeGroup);

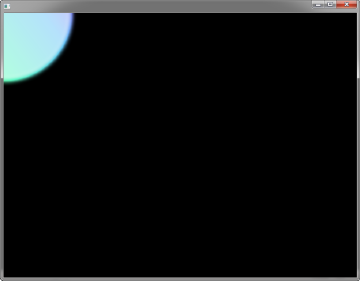
The group blendModeGroup sets up the structure for the overlay blend. The group contains two children. The first child is a new (unnamed) Group containing a new (unnamed) black rectangle and the previously created circles group. The second child is the previously created colors rectangle.

The setBlendMode() method applies the overlay blend to the colors rectangle. The final line of code adds the blendModeGroup to the scene graph as a child of the root node, as depicted in [Figure 7-2](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/animation.htm#BABBHDBC).

An overlay blend is a common effect in graphic design applications. Such a blend can darken an image or add highlights or both, depending on the colors in the blend. In this case, the linear gradient rectangle is used as the overlay. The black rectangle serves to keep the background dark, while the nearly transparent circles pick up colors from the gradient, but are also darkened.

[Figure 7-7](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/animation.htm#BABDJFAH) shows the results. You will see the full effect of the overlay blend when you animate the circles in the next step.

***Figure 7-7 Overlay Blend***



## **Add Animation**

The final step is to use JavaFX animations to move the circles:

1. If you have not done so already, add import static java.lang.Math.random; to the list of import statements.
2. Add the animation code in [Example 7-6](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/animation.htm#BABGIACI) before the primaryStage.show() line.

***Example 7-6 Animation***

Timeline timeline = new Timeline();

for (Node circle: circles.getChildren()) {

timeline.getKeyFrames().addAll(

new KeyFrame(Duration.ZERO, // set start position at 0

new KeyValue(circle.translateXProperty(), random() \* 800),

new KeyValue(circle.translateYProperty(), random() \* 600)

),

new KeyFrame(new Duration(40000), // set end position at 40s

new KeyValue(circle.translateXProperty(), random() \* 800),

new KeyValue(circle.translateYProperty(), random() \* 600)

)

);

}

// play 40s of animation

timeline.play();

Animation is driven by a timeline, so this code creates a timeline, then uses a for loop to add two key frames to each of the 30 circles. The first key frame at 0 seconds uses the properties translateXProperty and translateYProperty to set a random position of the circles within the window. The second key frame at 40 seconds does the same. Thus, when the timeline is played, it animates all circles from one random position to another over a period of 40 seconds.

[Figure 7-8](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/animation.htm#BABDAFAF) shows the 30 colorful circles in motion, which completes the application. For the complete source code, see the ColorfulCircles.java file, which is included in the downloadable [ColorfulCircles.zip](https://docs.oracle.com/javase/8/javafx/sample-apps/ColorfulCircles.zip) file..

***Figure 7-8 Animated Circles***



For More Read Below Links

* <https://docs.oracle.com/javase/8/javafx/visual-effects-tutorial/visual_effects.htm#JFXTE191>

**Rendering Bootstrap inside a JavaFX WebView**

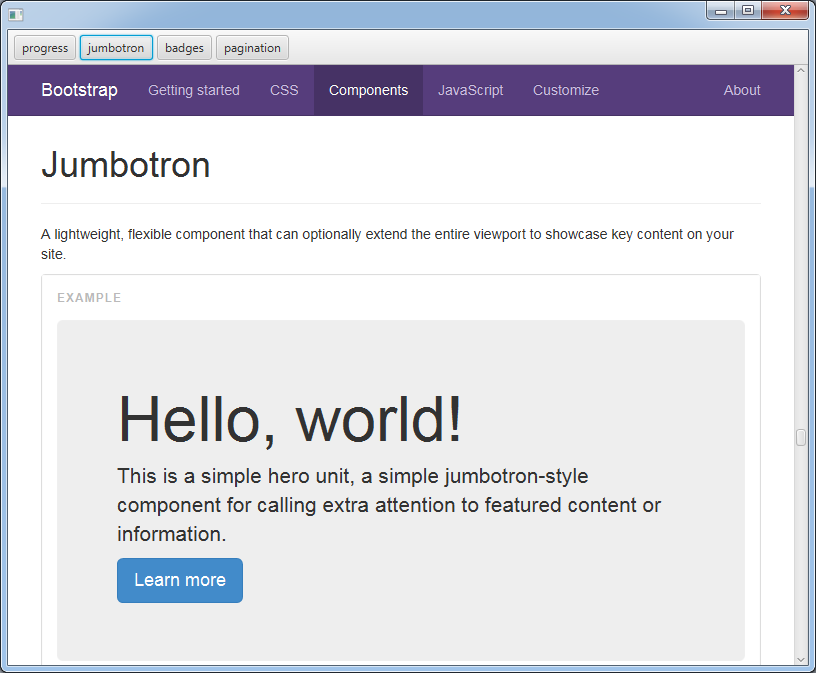
Bootstrap is an HTML based framework.

So to use Bootstrap in JavaFX, use JavaFX's HTML rendering component [WebView](http://docs.oracle.com/javafx/2/api/javafx/scene/web/WebView.html) to render Bootstrap HTML/CSS and JavaScript.

**Sample Application**

Sample application performing a basic integration of Bootstrap and a JavaFX UI.

The JavaFX buttons on the top of the screen navigate around a WebView page to render different kinds of Bootstrap components.



import javafx.application.Application;

import javafx.event.ActionEvent;

import javafx.event.EventHandler;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.ToolBar;

import javafx.scene.layout.VBox;

import javafx.scene.web.WebView;

import javafx.stage.Stage;

public class BaseJump extends Application {

private static final String BOOTSTRAP\_PREFIX = "http://getbootstrap.com/components/#";

private enum Anchor { progress, jumbotron, badges, pagination }

@Override public void start(Stage stage) throws Exception {

final WebView webview = new WebView();

final ToolBar nav = new ToolBar();

for (Anchor anchor : Anchor.values()) {

nav.getItems().add(

new NavButton(

anchor,

webview

)

);

}

VBox layout = new VBox();

layout.getChildren().addAll(

nav,

webview

);

Scene scene = new Scene(layout);

stage.setScene(scene);

stage.show();

}

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

private class NavButton extends Button {

public NavButton(final Anchor anchor, final WebView webview) {

setText(anchor.toString());

setOnAction(new EventHandler<ActionEvent>() {

@Override

public void handle(ActionEvent event) {

webview.getEngine().load(BOOTSTRAP\_PREFIX + anchor.toString());

}

});

}

}

}

**Additional, slightly unrelated question**

Is it possible to intercept button click events from a web view?

Yes. You can attach click handlers in Java code through the w3c DOM API access WebEngine provides. See the WebEngine documentation for details:

**Access to Document Model**

The WebEngine objects create and manage a Document Object Model (DOM) for their Web pages. The model can be accessed and modified using Java DOM Core classes. The getDocument() method provides access to the root of the model. Additionally DOM Event specification is supported to define event handlers in Java code.

The following example attaches a Java event listener to an element of a Web page. Clicking on the element causes the application to exit:

EventListener listener = new EventListener() {

public void handleEvent(Event ev) {

Platform.exit();

}

};

Document doc = webEngine.getDocument();

Element el = doc.getElementById("exit-app");

((EventTarget) el).addEventListener("click", listener, false);

However, for me often is is easier to handle interfacing with w3c documents using JavaScript (specifically jQuery), rather than Java. Here is an example of issuing from Java code, [a jQuery call to provide click handlers in a WebView](https://gist.github.com/jewelsea/3077942).

**On Fextile (Bootstrap look for native JavaFX controls)**

Why not just port bootstrap.css to conform to the javafx naming conventions?

Because:

* There is more to bootstrap than just the css, it's a full responsive UI framework with JavaScript based active controls and a user extension mechanism.
* Rendering in WebView will render the bootstrap html in JavaFX exactly the same as if it were in a web browser, so why port when you already have something which will work perfectly with no extra effort?
* It's a moving target, the bootstrap.css trunk project gets many contributions from hundreds of developers, it would be difficult to keep up with that with a home-grown JavaFX port, though if you selected just a smaller subset of bootstrap features keeping in sync would be easier.

Still, it is possible to do the port (as linked in Philippe's answer), and that is what Takayuki Okazaki has created in his [Fextile project](https://github.com/watermint/Fextile): "Twitter Bootstrap like UI framework for JavaFX. Apply themes to your application just like Twitter Bootstrap via JavaFX CSS.". Don't expect an exact match with bootstrap in HTML, but it should allow your JavaFX controls which don't use HTML to have a pretty close look to what you would achieve with bootstrap in HTML.

You could also create a hybrid application, where some parts of the UI are from HTML with bootstrap and some are rendered from JavaFX controls using Fextile. If you applied such an approach to the sample application in this answer, then the JavaFX buttons "progress", "jumbotron", etc. would look like their HTML bootstrap counterparts, giving the whole application a more consistent look and feel.

Also, note that there is a similar project for [Foundation styles for JavaFX](https://github.com/devork/golan), as announced in this [Oracle JavaFX forum post](https://community.oracle.com/thread/2490009?tstart=0). This project mimics the basic [Foundation](http://foundation.zurb.com/) look for native JavaFX controls. For some usages, adopting Foundation styles may be more appropriate than Bootstrap styles as the project is smaller in scope than Bootstrap (as far as I know).

Here is a Q&A (from the Oracle JavaFX forum post) on how to creating the Foundation style (so somebody can get a relative idea of what is involved for extending Fextile for additional Bootstrap style features). Note that the Q&A is a little old and since then a [CSS analyzer](https://blogs.oracle.com/jeromec/entry/using_the_new_css_analyzer) has been added to SceneBuilder:

1) How difficult was this work?

No at all: the whole experience was very pleasant and very easy to do. This is my first JavaFX app (a map style editor with real time preview)

2) Was it very time consuming?

No: using the preview ScenceBuilder 1.1 the styles are updated on the fly - the SceneBuilder could do with a built in CSS editor, but that's only minor: the workflow was quite simple anyway

3) For a simple port do you think that you need any design skills at all, or could anybody have really done this who knows a bit of css/html/javafx?

Anyone can do this: my background is server side code - I don't do much in the way of front ends - I know JS and HTML very well but my CSS leaves a lot to me desired: so basically if I can do it ...

4) Was the difference between the javafx css syntax and the html css syntax a major pain or not really an issue?

Once I got used to it, made no difference although I do keep forgetting to add '-fx-' and the -fx-text-fill I always type as -fx-text-color ...

5) Similarly did a lack of a one-to-one correspondence between html document tags (e.g. the header tags) and JavaFX complicate this?

No

6) Will the upcoming rich text support in JavaFX 8 simplify (or make possible) more of these kinds of ports?

I need to have a look at this: as I said I'm a complete beginner with JavaFX so I'm still catching up on the current implementation.

The bootstrap styles would be really nice: a lot of people who I've showed the app to are quite amazed when I tell them its Java and not an embedded web app.