Java Programming, 9e

Chapter 15

Using JavaFX and Scene Builder

Objectives (1 of 2)

- Describe JavaFX
- Describe the life cycle of a JavaFX application
- Recognize JavaFX structure: stage, scene, and widgets
- Write an application using JavaFX

Objectives (2 of 2)

- Create JavaFX applications using Scene Builder
- Use widgets as design elements in FXML layouts
- Use CSS to create visual effects
- Create animations in JavaFX

What is **JavaFX**? (1 of 3)

- Media and graphics framework that creates applications that use Graphical User Interfaces (GUI) in Java applications
  - **User Interface** (**UI**): Every screen that allows a user to interact with a program
- Lightweight
- **Hardware accelerated**
- Designed to replace `Swing`
  - Can be integrated into existing `Swing` applications
- Separates UI development from application logic

What is JavaFX? (2 of 3)

- JavaFX provides two tools to create interfaces:
  - **Scene Builder:** allows developer to create a UI visually
  - **Cascading Style Sheets** (**CSS**): style sheet language that describes presentation of a document

What is JavaFX? (3 of 3)

- **Markup language**
  - Used to design presentation, formatting, layout, and style of text
- **Declarative language**
  - High-level language
  - Defines desired result without explicitly listing commands to accomplish task
- **FXML**
  - XML-based declarative markup language used to define UIs
  - Used by JavaFX

The Life Cycle of JavaFX Applications (1 of 5)

- JavaFX applications extend from `Application` class
  - Must call `launch()` to launch an FX application
  - Other methods are called automatically when an application runs
    - `init()`: used for initialization tasks
    - `start()`: performs most of the work of an `Application`
    - `stop()`: executes when an application is finished

The Life Cycle of JavaFX Applications (2 of 5)

- A JavaFX application finishes when:
  - It calls `Platform.exit()`, or when
  - The last window in the application has been closed

The Life Cycle of JavaFX Applications (3 of 5)

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

public class LifeCycleDemo extends Application
{
   public static void main(String[] args)
   {
      launch(args);
   }
   @Override
   public void init()
   {
      System.out.println("In init() method");
   }
   @Override
   public void start(Stage primaryStage)
   {
      primaryStage.setTitle("Life cycle demo");
      StackPane root = new StackPane();
      primaryStage.setScene(new Scene(root, 300, 75));
      primaryStage.show();
      System.out.println("In start() method");
   }
   @Override
   public void stop()
   {
      System.out.println("In stop() method");
   }
}
```

Figure 15-1 LifeCycleDemo application

The Life Cycle of JavaFX Applications (4 of 5)

Figure 15-2 `LifeCycleDemo` application at start of run

The Life Cycle of JavaFX Applications (5 of 5)

```
In init() method
In start() method
In stop() method
```

Figure 15-3 LifeCycleDemo output at end of execution

Understanding JavaFX structure: Stage, Scene, Panes and Widgets (1 of 5)

- `Application`: provides entry point for a JavaFX application
- **`Stage`**: represents entire window in an application
  - Describes a container for an application
- **`Scene`**: holds content inside a window
  - Resides inside the `Stage`

Understanding JavaFX structure: Stage, Scene, Panes and Widgets (2 of 5)

- **`StackPane`**: controls the design layout of scenes
  - Type of container for layouts
  - Stacks pane layouts in a back to front order
  - **Root node**, or just **root**, is topmost layout
    - Is passed to the scene constructor

Understanding JavaFX structure: Stage, Scene, Panes and Widgets (3 of 5)

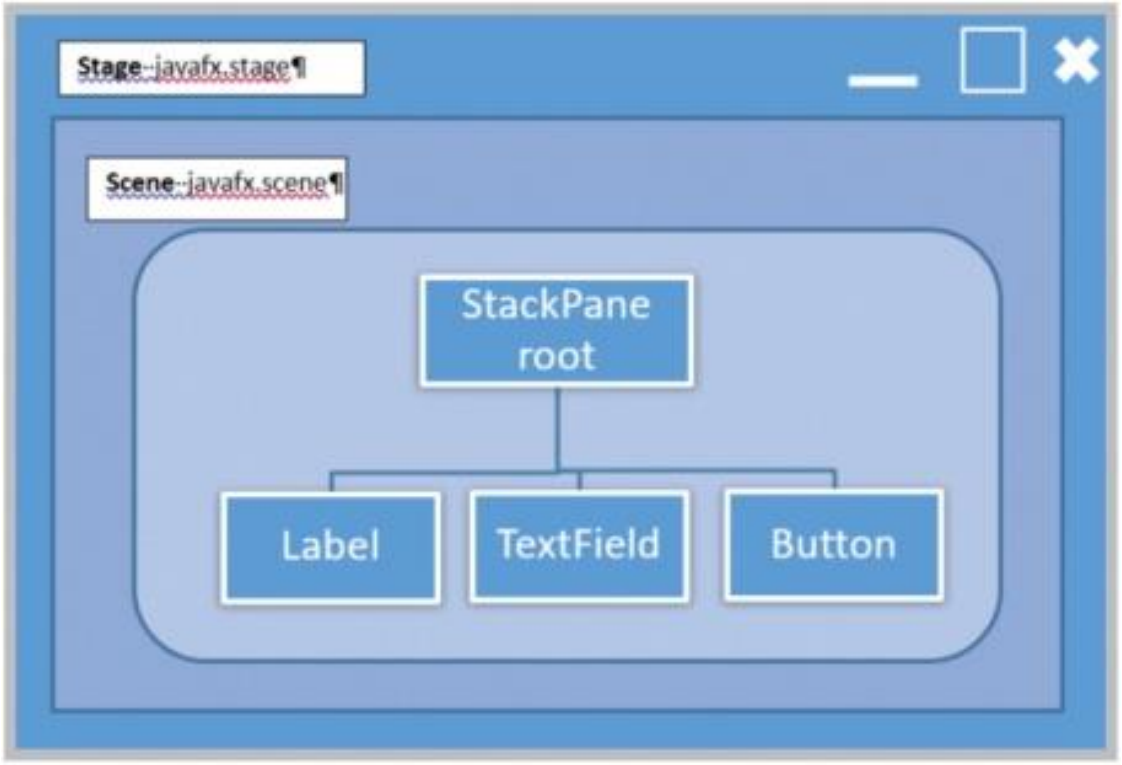

Figure 15-4 Relationship of Stage and Scene

Understanding JavaFX structure: Stage, Scene, Panes and Widgets (4 of 5)

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;


public class MyFirstJavaFXApp extends Application
{

    @Override
    public void start(Stage primaryStage)
    {
       Button btn = new Button();
       btn.setText("Click me");
       btn.setOnAction(new EventHandler<ActionEvent>()
               {
                   @Override
                   public void handle(ActionEvent event)
                   {
                       System.out.println("My First JavaFX App!!");
                   }
               });

        StackPane root = new StackPane();
        root.getChildren().add(btn);
        Scene scene = new Scene(root, 400, 300);

        primaryStage.setTitle("My First JavaFX App!!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
```

Figure 15-5 Code for MyFirstJavaFXApp application

Understanding JavaFX structure: Stage, Scene, Panes and Widgets (5 of 5)

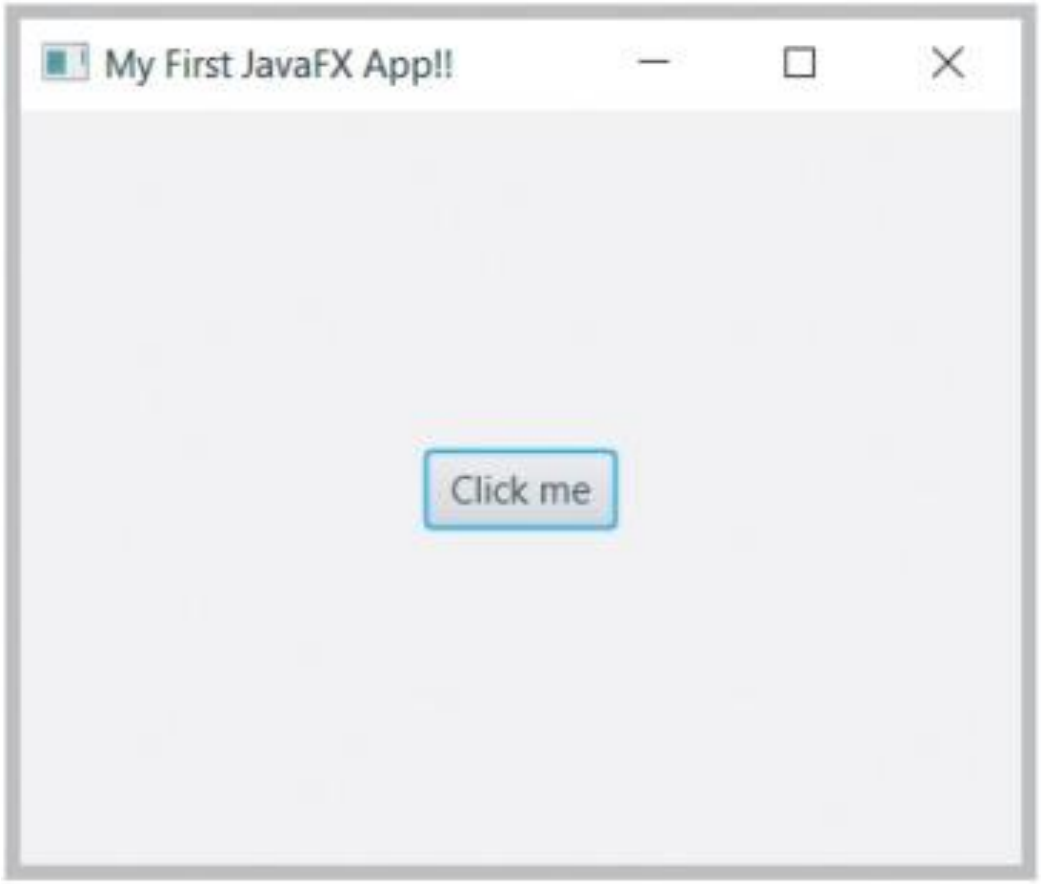

Figure 15-6 Output of MyFirstJavaFXApp application

Deploying JavaFX Applications

1. Run as a standalone program
   - For example: `java -jar MyApp.jar`
2. From a remote server, by clicking a link on a Web page
   - Application is downloaded
   - Then, desktop shortcut can be used to start the Web application
3. Embedded in a Web page
   - Application is hosted and runs on remote server
4. Self-contained application can be installed locally
   - Launched in same manner as other applications

Creating JavaFX applications using Scene Builder (1 of 2)

- Scene Builder
  - Add-in to Java
  - "**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et" (**WYSIWYG**) **drag and drop** capability
  - Automatically generates FXML code for layout
  - Allows for **live editing**
  - **Cross-platform development** for Windows, Linux, Mac OS, and the Web

Creating JavaFX applications using Scene Builder (2 of 2)

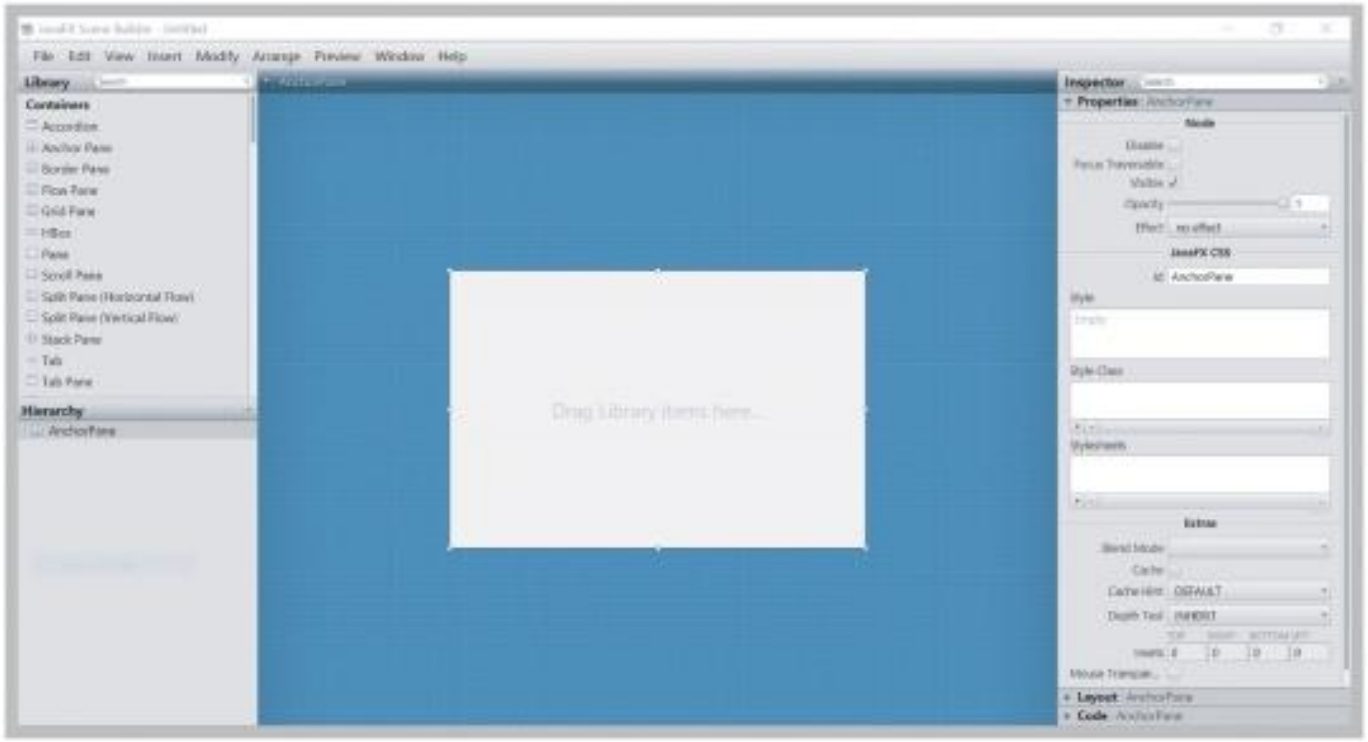

Figure 15-8 Scene Builder

Source: JavaFX Scene Builder

Scene Builder Sections (1 of 3)

- Menu Bar
  - Access to menu of commands
- Path, Selection, and Message Bar
  - Displays path to selected element
  - Can select a widget to put in focus
  - Error status
- Content Panel
  - Where you position widgets to create FXML layout
- Library Panel
  - Contains various widgets to create the layout

Scene Builder Sections (2 of 3)

- Document Panel
  - Contains the Hierarchy and Controller sections
- Controller Panel
  - Where you connect GUI to Java code
- Inspector Panel
  - Properties and Layout sections: set or modify properties
- CSS Analyzer Panel
  - Allows you to set the CSS properties

Scene Builder Sections (3 of 3)

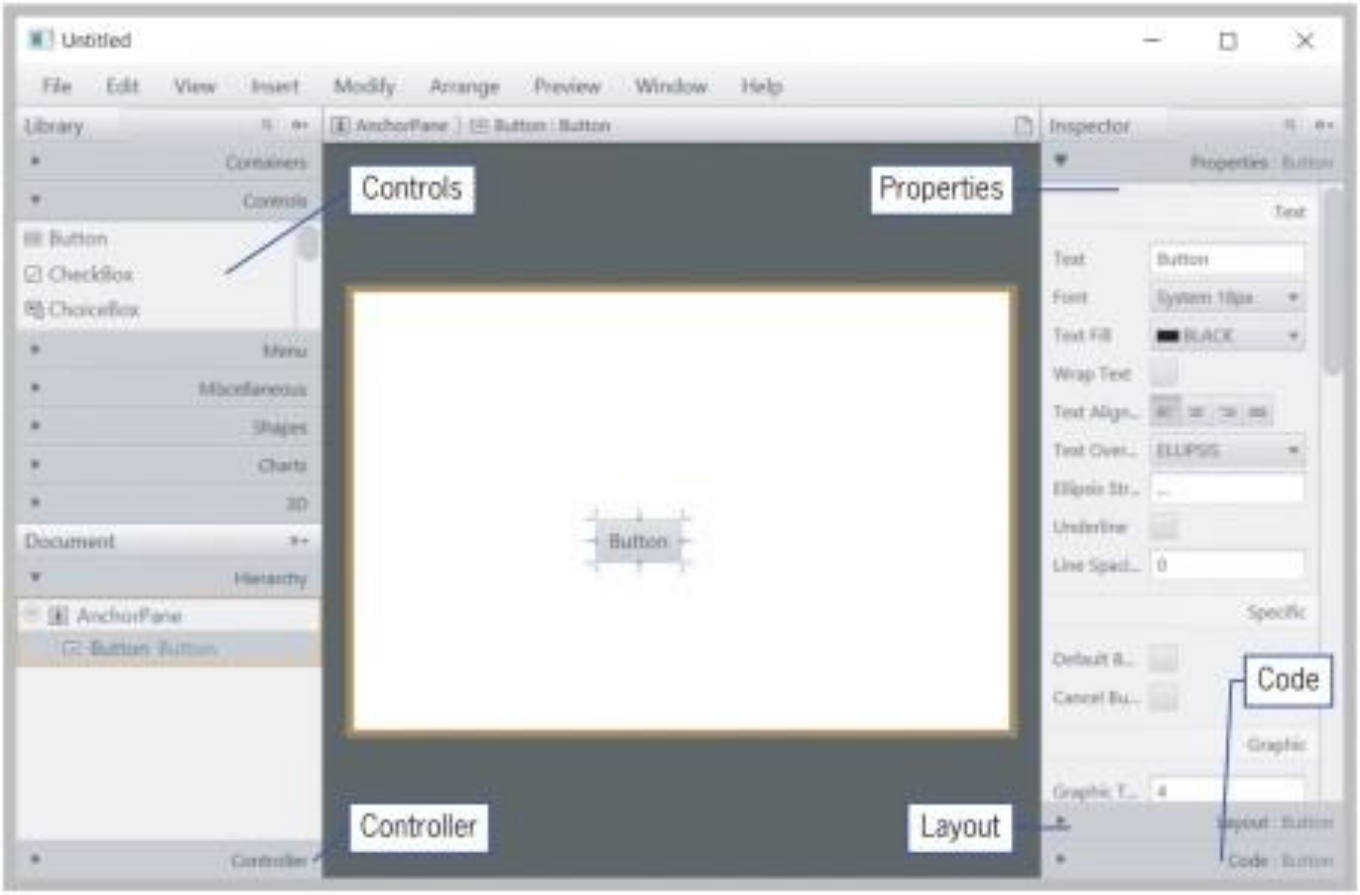

Figure 15-12 Scene Builder sections

Source: JavaFX Scene Builder

Using Widgets as Design Elements in FXML Layouts (1 of 3)

- Common widgets:
  - Text boxes, labels, radio buttons, check boxes, menu bars, and scroll bars
- Widgets in JavaFX are similar to those in `Swing`
  - `TextArea` similar to `TextFields`
    - But are multi-line to contain more text
  - Radio buttons and check boxes also use groups
    - But `ToggleGroups` are named in Properties panel in Scene Builder

Using Widgets as Design Elements in FXML Layouts (2 of 3)

- Content panel in Scene Builder is your design canvas
- Red alignment lines can be used to help with placement
- You can use the GUI to duplicate, delete, resize, and move widgets

Using Widgets as Design Elements in FXML Layouts (3 of 3)

- Some widgets can be re-ordered
  - `FlowPane`, `TextFlow`, `TilePane`, `ToolBar`, `HBox`, and `VBox` containers
  - A gray line appears to show placement or order location of selected control
- Many properties are associated with every widget
  - Sizes, padding, spacing, text and colors

Using CSS to Create Visual Effects (1 of 10)

- Scene Builder uses JavaFX **Modena** FX8 CSS style by default
- To use a different style
  - Customize it, or
  - Create your own CSS style

Using CSS to Create Visual Effects (2 of 10)

- Scene Builder does not generate CSS files
  - Use a CSS editor (e.g., your Java IDE)
- Use a style sheet on any level of your layout
  - CSS rules on a parent control are inherited by children widgets

Using CSS to Create Visual Effects (3 of 10)

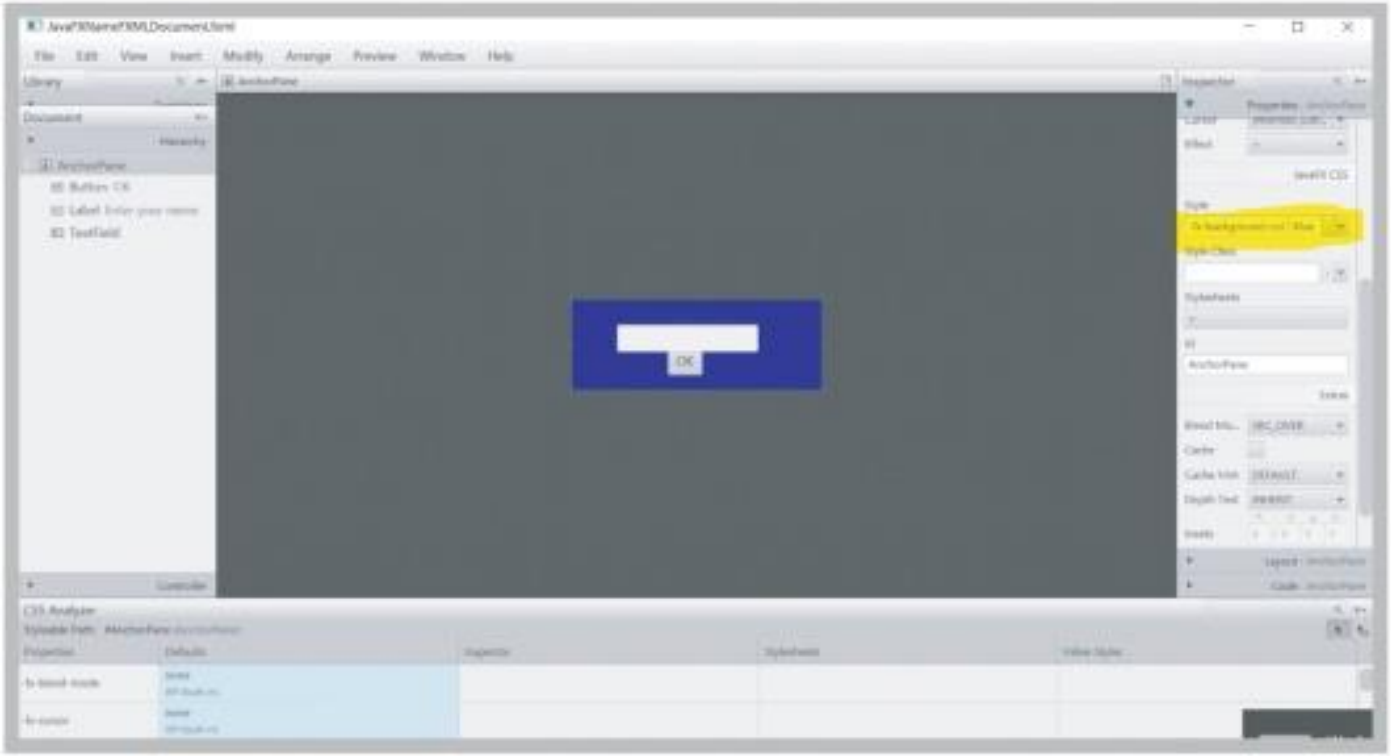

Figure 15-14 Individual CSS control edit

Source: JavaFX Scene Builder

Using CSS to Create Visual Effects (4 of 10)

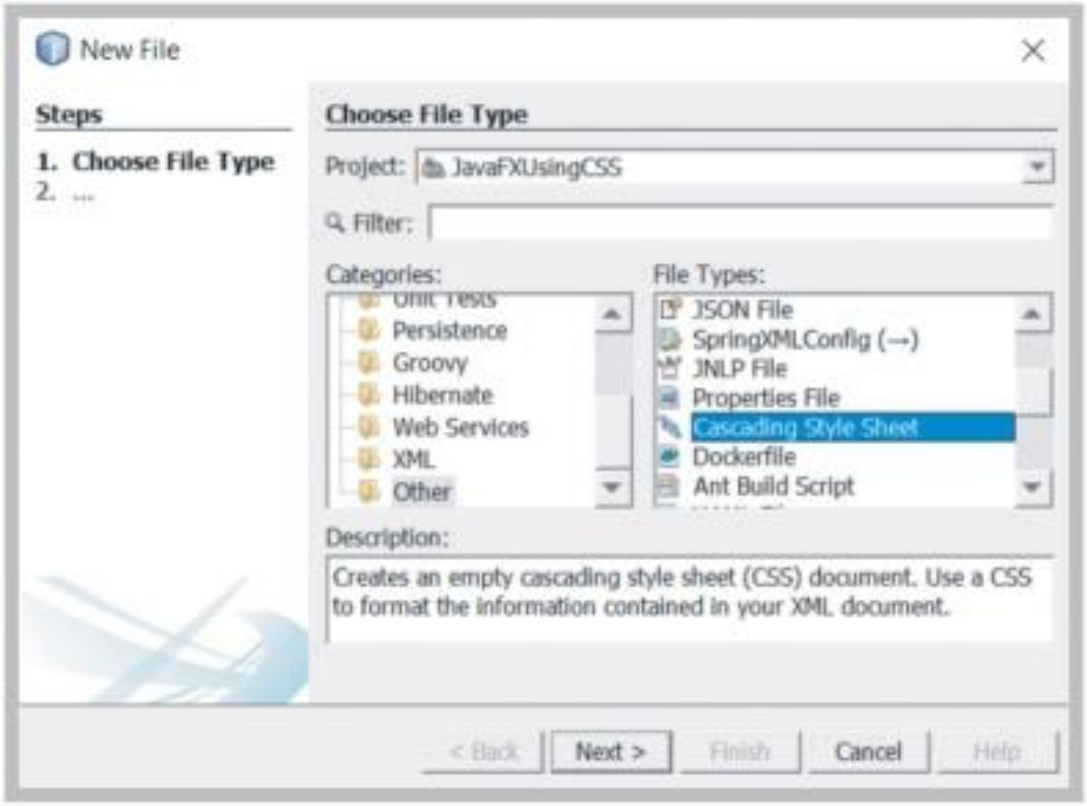

Figure 15-15 Adding a CSS file from the IDE

Source: NetBeans IDE

Using CSS to Create Visual Effects (5 of 10)

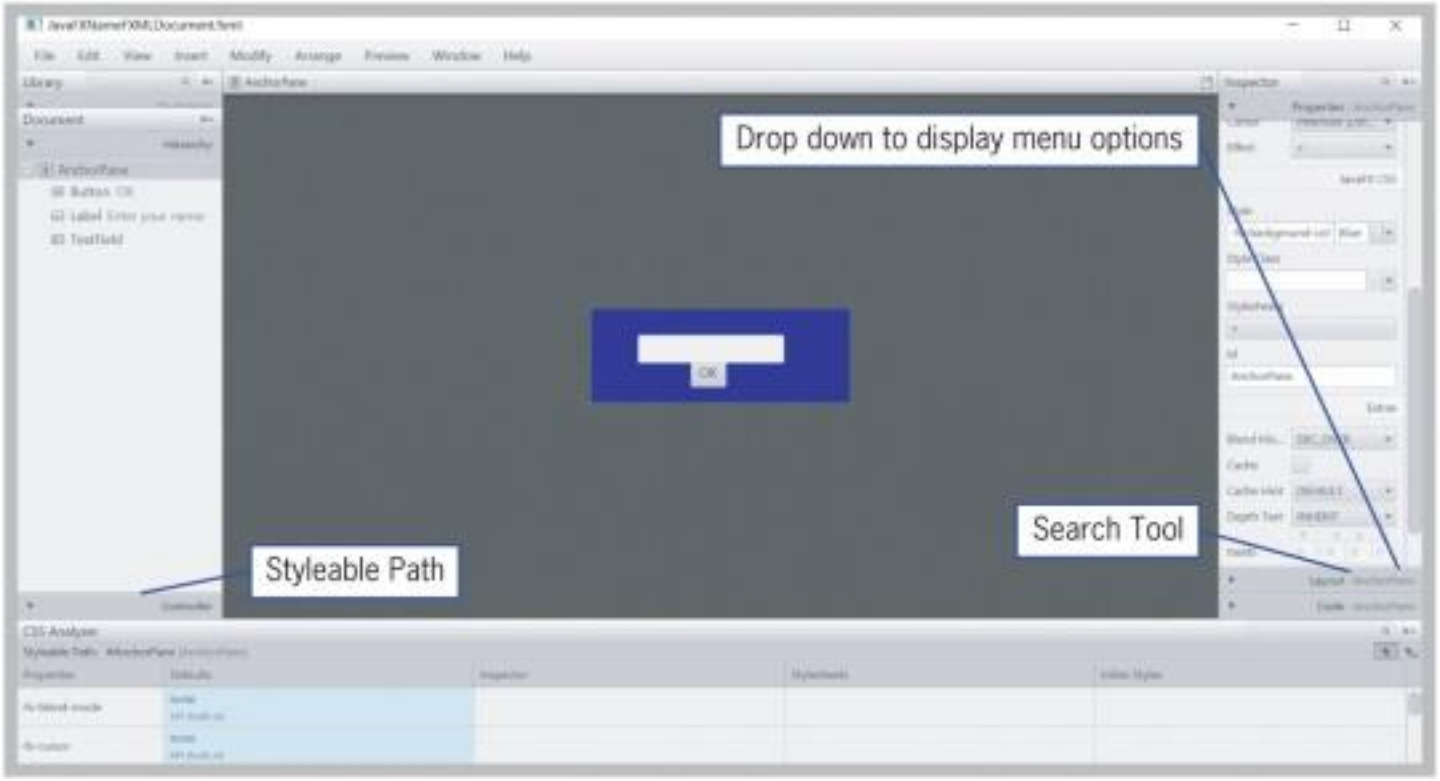

Figure 15-16 CSS Analyzer Pane Opened

Source: JavaFX Scene Builder

Using CSS to Create Visual Effects (6 of 10)

- **CSS Analyzer Menu** options:
  - **View As**
    - Choices are **Table** (default), **Rules**, or **Text**
  - **Copy Styleable Path**
    - Used to copy value in Styleable Path text field (and later pasted into CSS file)
  - **Hide/Show Properties with Default Values**
    - Used to show or hide properties that have default values

Using CSS to Create Visual Effects (7 of 10)

- **CSS Analyzer Menu** options (continued):
  - **Join/Split Defaults**
    - Allows you to Join (default) or Split the Defaults column
    - Join: Column displays values set as a single column **Defaults**
    - Split: Displays as two separate columns, **API Defaults** and **FX Theme Defaults**

Using CSS to Create Visual Effects (8 of 10)

- **Styleable Path** text field
  - To copy path using CSS Analyzer menu
  - Then paste path in CSS file to assign a new style value to controls
- **CSS Picking Mode** button
  - To select a widget on the Content panel

Using CSS to Create Visual Effects (9 of 10)

- A table with five columns is displayed beneath the Styleable Path
  1. **Properties**
  2. **Defaults**
  3. **Inspector**
  4. **Stylesheets**
  5. **Inline Styles**
- Cog icon in cell used to select **Reveal in Inspector** to display the Style text view in Inspector panel

Using CSS to Create Visual Effects (10 of 10)

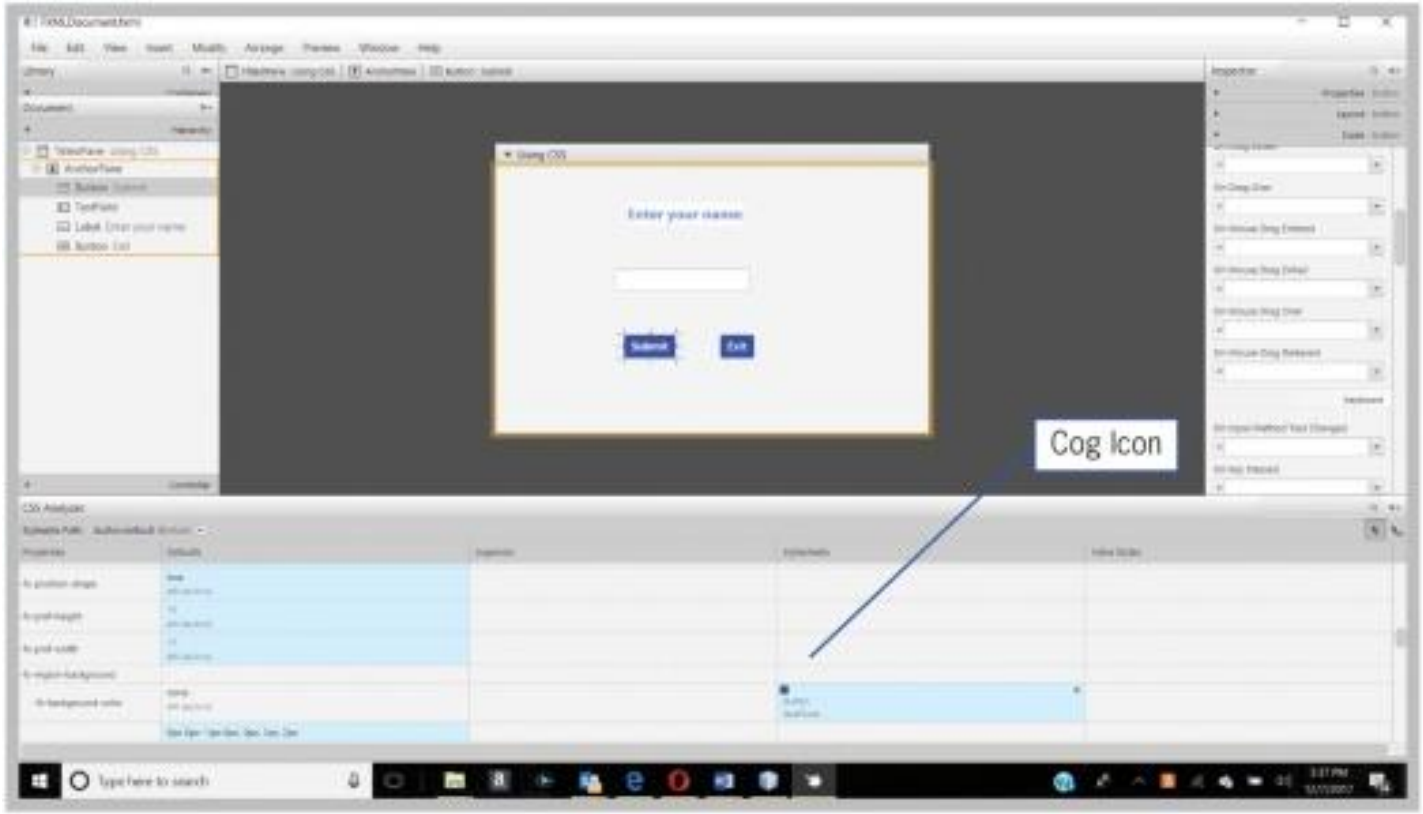

Figure 15-17 CSS Analyzer panel with cog icon

Source: JavaFX Scene Builder

Creating Animations in JavaFX (1 of 3)

- **Transition**
  - Change of some kind, such as in size, scale, color, fade, or position
  - **Parallel transition:** executes multiple transitions concurrently
  - **Sequential transition:** executes transitions one after another

Creating Animations in JavaFX (2 of 3)

- **Timeline transitions**
  - Update property values along the progression of time
  - Similar to creating an animation using frames
  - **Key frame animations:** use start and end frames called key frames (snapshots)

Creating Animations in JavaFX (3 of 3)

- **Interpolation**
  - Process where movement positions of an animated object are calculated between the start and the end points of the object
  - JavaFX has an interpolator built in
  - You can write your own custom interpolator

Don’t Do It

- Don’t forget to run the Make Controller command any time you delete a widget from the Scene Builder’s Content panel
- Don’t forget to add a semicolon at the end of any CSS file you edit

Summary (1 of 2)

- JavaFX is a powerful media and graphics framework that is used to create applications that use Graphical User Interfaces (GUI) in Java applications
- JavaFX applications extend from the `Application` class, which extends directly from the `Object` class
- The `Stage` class describes a container for an application; in a JavaFX application, a `Stage` object represents the entire window
- JavaFX Scene Builder provides a visual layout environment that lets you design the UI for JavaFX applications without needing to write any code
- Commonly used widgets include text boxes, labels, radio buttons, check boxes, menu bars, and scroll bars

Summary (2 of 2)

- CSS allows the developer to customize the appearance of a layout using fonts, styles, colors, and different effects on the appearance of the layout and its widgets
- Animation in JavaFX can be as simple as fading one scene to another, rotating an object, or having an object follow a path, and is created by modifying an object's properties such as color, size, opacity and location