# 22.2 Controlling Fonts with Cascading Style Sheets (CSS)

In Chapters 12–13, you built JavaFX GUIs using Scene Builder. You specified a particular JavaFX object's appearance by selecting the object in Scene Builder, then setting its property values in the **Properties** inspector. With this approach, if you want to change the GUI's appearance, you must edit each object. If you have a large GUI in which you want to make the same changes to multiple objects, this can be time consuming and error prone.

In this chapter, we format JavaFX objects using a technology called **Cascading Style Sheets (CSS)** that's typically used to style the elements in web pages. CSS allows you to specify *presentation* (e.g., fonts, spacing, sizes, colors, positioning) separately from the GUI's *structure* and *content* (layout containers, shapes, text, GUI components, etc.). If a JavaFX GUI's presentation is determined entirely by CSS rules, you can simply swap in a new style sheet to change the GUI's appearance.

In this section, you'll use CSS to specify the font properties of several `Label`s and the spacing and padding properties for the `VBox` layout that contains the `Label`s. You'll place **CSS**

**rules** that specify the font properties, spacing and padding in a separate file that ends with the `.css` **filename extension**, then reference that file from the FXML. As you'll see,

- before referencing the CSS file from the FXML, Scene Builder displays the GUI without styling, and
- after referencing the CSS file from the FXML, Scene Builder renders the GUI with the CSS rules applied to the appropriate objects.

For a complete reference that shows

- all the JavaFX CSS properties,
- the JavaFX `Node` types to which the attributes can be applied, and
- the allowed values for each attribute

visit:

```
https://docs.oracle.com/javase/8/javafx/api/javafx/sc
```

# 22.2.1 CSS That Styles the GUI

Figure 22.1 presents this app's CSS rules that specify the `VBox`'s and each `Label`'s style. This file is located in the same folder as the rest of the example's files.

# `.vbox` CSS Rule—Style Class Selectors

Lines 4–7 define the `.vbox` CSS rule that will be applied to this app's `VBox` object (lines 8–18 of Fig. 22.2). Each CSS rule begins with a **CSS selector** which specifies the JavaFX objects that will be styled according to the rule. In the `.vbox` CSS rule, `.vbox` is a **style class selector**. The CSS properties in this rule are applied to any JavaFX object that has a `styleClass` property with the value `"vbox"`. In CSS, a style class selector begins with a dot (`.`) and is followed by its **class name** (not to be confused with a Java class). By convention, selector names typically have all lowercase letters, and multi-word names separate each word from the next with a dash (`-`).

```
1  /* Fig. 22.1: FontsCSS.css */
2  /* CSS rules that style the VBox and Labels */
3
4  .vbox {
5     -fx-spacing: 10;
6     -fx-padding: 10;
7  }
8
9  #label1 {
10    -fx-font: bold 14pt Arial;
11 }
12
13 #label2 {
14    -fx-font: 16pt "Times New Roman";
15 }
16
17 #label3 {
```

```
18      -fx-font: bold italic 16pt "Courier New";
19   }
20
21   #label4 {
22      -fx-font-size: 14pt;
23      -fx-underline: true;
24   }
25
26   #label5 {
27      -fx-font-size: 14pt;
28   }
29
30   #label5 .text {
31      -fx-strikethrough: true;
32   }
```

# Fig. 22.1

CSS rules that style the `VBox` and `Label`s.

Each CSS rule's body is delimited by a set of required braces (`{}`) containing the CSS properties that are applied to objects matching the CSS selector. Each JavaFX CSS property name begins with `-fx-`[1] followed by the name of the corresponding JavaFX object's property in all lowercase letters. So, `-fx-spacing` in line 5 of Fig. 22.1 defines the value for a JavaFX object's `spacing` property, and `-fx-padding` in line 6 defines the value for a JavaFX object's `padding` property. The value of each property is specified to the right of the required colon (`:`). In this case, we set `-fx-spacing` to `10` to place 10 pixels of vertical space between objects in the

`VBox`, and `-fx-padding` to `10` to separate the `VBox`'s contents from the `VBox`'s edges by 10 pixels at the top, right, bottom and left edges. You also can specify the `-fx-padding` with four values separated by spaces. For example,

1. According to the JavaFX CSS Reference Guide at `https://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html`, JavaFX CSS property names are designed to be processed in style sheets that may also contain HTML CSS. For this reason, JavaFX's CSS property names are prefixed with "`-fx-`" to ensure that they have distinct names from their HTML CSS counterparts.

```
   -fx-padding: 10 5 10 5
```

specifies 10 pixels for the top padding, 5 for the right, 10 for the bottom and 5 for the left. We show how to apply the `.vbox` CSS rule to the `VBox` object in Section 22.2.2.

# `#label1` CSS Rule—ID Selectors

Lines 9–11 define the `#label1` CSS rule. Selectors that begin with # are known as **ID selectors**—they are applied to objects with the specified ID. In this case, the `#label1` selector matches the object with the `fx:id label1`—that is, the `Label` object in line 12 of Fig. 22.2. The `#label1` CSS rule specifies the CSS property

```
-fx-font: bold 14pt Arial;
```

This rule sets an object's `font` property. The object to which

this rule applies displays its text in a bold, 14-point, Arial font. The `-fx-font` property can specify all aspects of a font, including its style, weight, size and font family—the size and font family are required. There are also properties for setting each font component: `-fx-font-style`, `-fx-font-weight`, `-fx-font-size` and `-fx-font-family`. These are applied to a JavaFX object's similarly named properties. For more information on specifying CSS font attributes, see

```
https://docs.oracle.com/javase/8/javafx/api/javafx/sc
```

For a complete list of CSS selector types and how you can combine them, see

```
https://www.w3.org/TR/css3-selectors/
```

# `#label2` CSS Rule

Lines 13–15 define the `#label2` CSS rule that will be applied to the `Label` with the `fx:id label2`. The CSS property

```
-fx-font: 16pt "Times New Roman";
```

specifies only the required font size (`16pt`) and font family

(`"Times New Roman"`) components—font family names with multiple words must be enclosed in double quotes.

## `#label3` CSS Rule

Lines 17–19 define the `#label3` CSS rule that will be applied to the `Label` with the `fx:id label3`. The CSS property

```
   -fx-font: bold italic 16pt "Courier New";
```

specifies all the font components—weight (`bold`), style (`italic`), size (`16pt`) and font family (`"Courier New"`).

## `#label4` CSS Rule

Lines 21–24 define the `#label4` CSS rule that will be applied to the `Label` with the `fx:id label4`. The CSS property

```
   -fx-font-size: 14pt;
```

specifies the font size `14pt`—all other aspects of this `Label`'s font are inherited from the `Label`'s parent container. The CSS property

```
   -fx-underline: true;
```

indicates that the text in the `Label` should be *underlined*—the default value for this property is `false`.

## `#label5` CSS Rule

Lines 26–28 define the `#label5` CSS rule that will be applied to the `Label` with the `fx:id label5`. The CSS property

```
   -fx-font-size: 14pt;
```

specifies the font size `14pt`.

## `#label5 .text` CSS Rule

Lines 30–32 define the `#label5 .text` CSS rule that will be applied to the `Text` object within the `Label` that has the `fx:id` value `"label5"`. The selector in this case is a combination of an ID selector and a style class selector. Each `Label` contains a `Text` object with the CSS class `.text`. When applying this CSS rule, JavaFX first locates the object with the ID `label5`, then within that object looks for a nested object that specifies the class `text`.

The CSS property

```
-fx-strikethrough: true;
```

indicates that the text in the `Label` should be displayed with a line through it—the default value for this property is `false`.

# 22.2.2 FXML That Defines the GUI—Introduction to XML Markup[2]

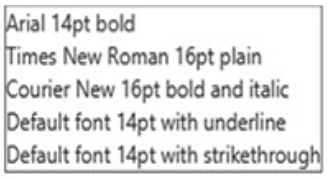
2. In many of this chapter's examples, after creating a GUI in Scene Builder, we used a text editor to format the FXML, remove unnecessary properties that were inserted by Scene Builder and properties that we specified via CSS rules. For this reason, when you build these examples from scratch, your FXML may differ from what's shown in this chapter. You also can set a property to its default value in Scene Builder to remove it from the FXML.

Figure 22.2 shows the contents of `FontCSS.fxml`—the `FontCSS` app's FXML GUI, which consists of a `VBox` layout element (lines 8–18) containing five `Label` elements (lines 12–16). When you first drag five `Label`s onto the `VBox` and configure their text (Fig. 22.2(a)), all the `Label`s initially have the same appearance in Scene Builder. Also, initially there's no spacing between and around the `Label`s in the `VBox`.

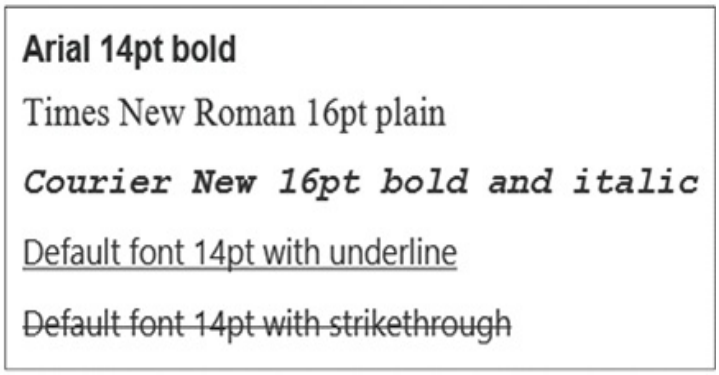
```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- Fig. 22.2: FontCSS.fxml -->
3  <!-- FontCSS GUI that is styled via external CSS
```

```
4
5   <?import javafx.scene.control.Label?>
6   <?import javafx.scene.layout.VBox?>
7
8   <VBox styleClass="vbox" stylesheets="@FontCSS.css
9       xmlns="http://javafx.com/javafx/8.0.60"
10      xmlns:fx="http://javafx.com/fxml/1">
11      <children>
12          <Label fx:id="label1" text="Arial 14pt bol
13          <Label fx:id="label2" text="Times New Roma
14          <Label fx:id="label3" text="Courier New 16
15          <Label fx:id="label4" text="Default font 1
16          <Label fx:id="label5" text="Default font 1
17      </children>
18  </VBox>
```

a) GUI as it appears in Scene Builder *before* referencing the completed CSS file




b) GUI as it appears in Scene Builder *after* referencing the `FontCSS.css` file containing the rules that style the `VBox` and the `Label`s




c) GUI as it appears in the *running* application


Fig. 22.2

`FontCSS` GUI that is styled via external CSS.

Description

# XML Declaration

Each FXML document begins with an **XML declaration** (line

1), which must be the first line in the file and indicates that the document contains XML markup. For FXML documents, line 1 must appear as shown in Fig. 22.2. The XML declaration's `version` **attribute** specifies the XML syntax version (`1.0`) used in the document. The `encoding` **attribute** specifies the format of the document's character—XML documents typically contain Unicode characters in `UTF-8` format (`https://en.wikipedia.org/wiki/UTF-8`).

# Attributes

Each XML attribute has the format

```
name="value"
```

The *name* and *value* are separated by `=` and the *value* placed in quotation marks (`""`). Multiple $name = value$ pairs are separated by whitespace.

# Comments

Lines 2–3 are XML comments, which begin with `<!--` and end with `-->`, and can be placed almost anywhere in an XML document. XML comments can span to multiple lines.

# FXML `import` Declarations

Lines 5–6 are **FXML `import` declarations** that specify the fully qualified names of the JavaFX types used in the document. Such declarations are delimited by `<?import` and `?>`.

# Elements

XML documents contain **elements** that specify the document's structure. Most elements are delimited by a **start tag** and an **end tag**:

- A start tag consists of **angle brackets** (`<` and `>`) containing the element's name followed by zero or more attributes. For example, the `VBox` element's start tag (lines 8–10) contains four attributes.
- An end tag consists of the element name preceded by a **forward slash** (`/`) in angle brackets—for example, `</VBox>` in line 18.

An element's start and end tags enclose the element's contents. In this case, lines 11–17 declare other elements that describe the `VBox`'s contents. Every XML document must have exactly one **root element** that contains all the other elements. In Fig. 22.2, `VBox` is the root.

A layout element always contains a `children` element (lines 11–17) containing the child `Node`s that are arranged by that layout. For a `VBox`, the `children` element contains the child `Node`s in the order they're displayed on the screen from top to bottom. The elements in lines 12–16 represent the `VBox`'s five

`Label`s. These are **empty elements** that use the shorthand start-tag-only notation:

```
<ElementName attributes />
```

in which the empty element's start tag ends with `/>` rather than `>`. The empty element:

```
<Label fx:id="label1" text="Arial 14pt bold" />
```

is equivalent to

```
<Label fx:id="label1" text="Arial 14pt bold">
</Label>
```

which does not have content between the start and end tags. Empty elements often have attributes (such as `fx:id` and `text` for each `Label` element).

# XML Namespaces

In lines 9–10, the `VBox` attributes

```
xmlns="http://javafx.com/javafx/8.0.60"
xmlns:fx="http://javafx.com/fxml/1"
```

specify the XML namespaces used in FXML markup. An XML **namespace** specifies a collection of element and attribute names that you can use in the document. The attribute

```
xmlns="http://javafx.com/javafx/8.0.60"
```

specifies the default namespace. FXML `import` declarations (like those in lines 5–6) add names to this namespace for use in the document. The attribute

```
xmlns:fx="http://javafx.com/fxml/1"
```

specifies JavaFX's `fx` namespace. Elements and attributes from this namespace (such as the `fx:id` attribute) are used internally by the `FXMLLoader` class. For example, for each FXML element that specifies an `fx:id`, the `FXMLLoader` initializes a corresponding variable in the controller class. The `fx:` in `fx:id` is a **namespace prefix** that specifies the namespace (`fx`) that defines the attribute (`id`). Every element or attribute name in Fig. 22.2 that does not begin with `fx:` is part of the default namespace.

# 22.2.3 Referencing the CSS File from FXML

For the `Label`s to appear with the fonts shown in Fig.

22.2(b), we must reference the `FontCSS.css` file from the FXML. This enables Scene Builder to apply the CSS rules to the GUI. To reference the CSS file:

1. Select the `VBox` in the Scene Builder.
2. In the **Properties** inspector, click the + button under the **Stylesheets** heading.
3. In the dialog that appears, select the `FontCSS.css` file and click **Open**.

This adds the `stylesheets` attribute (line 8)

```
   stylesheets="@FontCSS.css"
```

to the `VBox`'s opening tag (lines 8–10). The `@` symbol—called the local resolution operator in FXML—indicates that the file `FontCSS.css` is located relative to the FXML file on disk. No path information is specified here, so the CSS file and the FXML file must be in the same folder.

# 22.2.4 Specifying the VBox's Style Class

The preceding steps apply the font styles to the `Label`s, based on their ID selectors, but do not apply the spacing and padding to the `VBox`. Recall that for the `VBox` we defined a CSS rule using a *style class selector* with the name `.vbox`. To apply the CSS rule to the `VBox`:

1. Select the `VBox` in the Scene Builder.
2. In the **Properties** inspector, under the **Style Class** heading, specify the value `vbox` *without* the dot, then press *Enter* to complete the setting.

This adds the `styleClass` attribute

```
   styleClass="vbox"
```

to the `VBox`'s opening tag (line 8). At this point the GUI appears as in Fig. 22.2(b). You can now run the app to see the output in Fig. 22.2(c).

# 22.2.5 Programmatically Loading CSS

In the `FontCSS` app, the FXML referenced the CSS style sheet directly (line 8). It's also possible to load CSS files dynamically and add them to a `Scene`'s collection of style sheets. You might do this, for example, in an app that enables users to choose their preferred look-and-feel, such as a light background with dark text vs. a dark background with light text.

To load a stylesheet dynamically, add the following statement to the `Application` subclass's `start` method:

```
   scene.getStylesheets().add(
      getClass().getResource("FontCSS.css").toExternalFo
```

In the preceding statement:

- Inherited `Object` method `getClass` obtains a `Class` object representing the app's `Application` subclass.
- `Class` method `getResource` returns a `URL` representing the location of the file `FontCSS.css`. Method `getResource` looks for the file in the same location from which the `Application` subclass was loaded.
- `URL` method `toExternalForm` returns the `URL`'s `String` representation. This is passed to the `add` method of the `Scene`'s collection of style sheets—this adds the style sheet to the scene.