22.3 Displaying Two-Dimensional Shapes

JavaFX has two ways to draw shapes:

- You can define `Shape` and `Shape3D` (package `javafx.scene.shape`) subclass objects, add them to a container in the JavaFX stage and manipulate them like other JavaFX `Node`s.
- You can add a `Canvas` object (package `javafx.scene.canvas`) to a container in the JavaFX stage, then draw on it using various `GraphicsContext` methods.

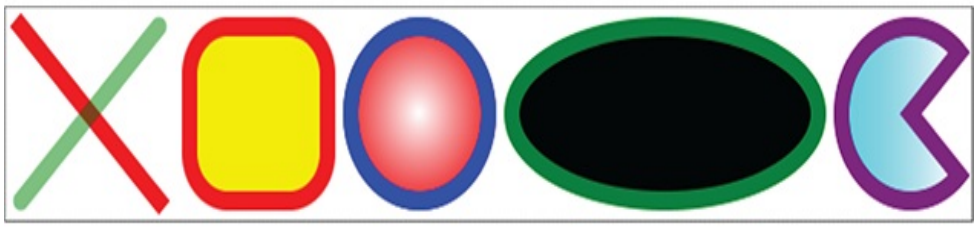
The `BasicShapes` example presented in this section shows you how to display two-dimensional `Shape`s of types `Line`, `Rectangle`, `Circle`, `Ellipse` and `Arc`. Like other `Node` types, you can drag shapes from the Scene Builder **Library**'s **Shapes** category onto the design area, then configure them via the **Inspector**'s **Properties**, **Layout** and **Code** sections—of course, you also may create objects of any JavaFX `Node` type programmatically.

22.3.1 Defining Two-Dimensional Shapes with FXML

Figure 22.3 shows the completed FXML for the `BasicShapes` app, which references the `BasicShapes.css` file (line 13) that we present in Section 22.3.2. For this app we dragged two `Line`s, a `Rectangle`, a `Circle`, an `Ellipse` and an `Arc` onto a `Pane` layout and configured their dimensions and positions in Scene Builder.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- Fig. 22.3: BasicShapes.fxml -->
3  <!-- Defining Shape objects and styling via CSS -
4
5  <?import javafx.scene.layout.Pane?>
6  <?import javafx.scene.shape.Arc?>
7  <?import javafx.scene.shape.Circle?>
8  <?import javafx.scene.shape.Ellipse?>
9  <?import javafx.scene.shape.Line?>
10 <?import javafx.scene.shape.Rectangle?>
11
12 <Pane id="Pane" prefHeight="110.0" prefWidth="63
13    stylesheets="@BasicShapes.css" xmlns="http://
14    xmlns:fx="http://javafx.com/fxml/1">
15       <children>
16       <Line fx:id="line1" endX="100.0" endY="100
17          startX="10.0" startY="10.0" />
18       <Line fx:id="line2" endX="10.0" endY="100.
19          startX="100.0" startY="10.0" />
20       <Rectangle fx:id="rectangle" height="90.0"
21          layoutY="10.0" width="90.0" />
22       <Circle fx:id="circle" centerX="270.0" cen
23          radius="45.0" />
24       <Ellipse fx:id="ellipse" centerX="430.0" c
25          radiusX="100.0" radiusY="45.0" />
26       <Arc fx:id="arc" centerX="590.0" centerY="
27          radiusX="45.0" radiusY="45.0" startAngl
28       </children>
29    </Pane>
```

a) GUI in Scene Builder with CSS applied—Ellipse's image fill does not show.



b) GUI in running app—Ellipse's image fill displays correctly.

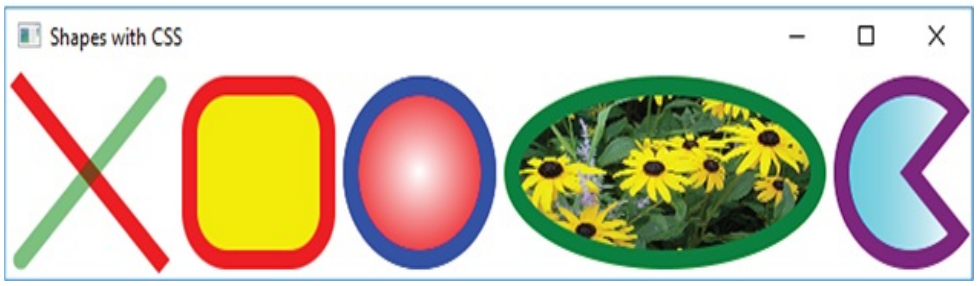


Fig. 22.3

Defining `Shape` objects and styling via CSS.

Description

For each property you can set in Scene Builder, there is a corresponding attribute in FXML. For example, the `Pane` object's **Pref Height** property in Scene Builder corresponds to the `prefHeight` attribute (line 12) in FXML. When you build this GUI in Scene Builder, use the FXML attribute values shown in Fig. 22.3. Note that as you drag each shape onto your design, Scene Builder automatically configures certain properties, such as the **Fill** and **Stroke** colors for the `Rectangle`, `Circle`, `Ellipse` and `Arc`. For each such property that does not have a corresponding attribute shown in Fig. 22.3, you can remove the attribute either by setting the

property to its default value in Scene Builder or by manually editing the FXML.

Lines 6–10 import the shape classes used in the FXML. We also specified `fx:id` values (lines 16 and 18) for the two `Line`s—we use these values in CSS rules with ID selectors to define separate styles for each `Line`. We removed the shapes' `fill`, `stroke` and `strokeType` properties that Scene Builder autogenerated. The default `fill` for a shape is black. The default stroke is a one-pixel black line. The default `strokeType` is centered—based on the stroke's thickness, half the thickness appears inside the shape's bounds and half outside. You also may display a shape's stroke completely inside or outside the shape's bounds. We specify the strokes and fills with the styles in Section 22.3.2.

`Line` Objects

Lines 16–17 and 18–19 define two `Line`s. Each connects two endpoints specified by the properties `startX`, `startY`, `endX` and `endY`. The *x*- and *y*-coordinates are measured from the top-left corner of the `Pane`, with *x*-coordinates increasing left to right and *y*-coordinates increasing top to bottom. If you specify a `Line`'s `layoutX` and `layoutY` properties, then the `startX`, `startY`, `endX` and `endY` properties are measured from that point.

`Rectangle` Object

Lines 20–21 define a `Rectangle` object. A `Rectangle` is displayed based on its `layoutX`, `layoutY`, `width` and `height` properties:

- A `Rectangle`'s upper-left corner is positioned at the coordinates specified by the `layoutX` and `layoutY` properties, which are inherited from class `Node`.
- A `Rectangle`'s dimensions are specified by the `width` and `height` properties—in this case they have the same value, so the `Rectangle` defines a square.

`Circle` Object

Lines 22–23 define a `Circle` object with its center at the point specified by the `centerX` and `centerY` properties. The `radius` property determines the `Circle`'s size (two times the `radius`) around its center point.

`Ellipse` Object

Lines 24–25 define an `Ellipse` object. Like a `Circle`, an `Ellipse`'s center is specified by the `centerX` and `centerY` properties. You also specify `radiusX` and `radiusY` properties that help determine the `Ellipse`'s width (left and right of the center point) and height (above and below the center point).

`Arc` Object

Lines 26–27 define an `Arc` object. Like an `Ellipse`, an `Arc`'s center is specified by the `centerX` and `centerY` properties, and the `radiusX` and `radiusY` properties determine the `Arc`'s width and height. For an `Arc`, you also specify:

- `length`—The arc's length in degrees (0–360). Positive values sweep counterclockwise.
- `startAngle`—The angle in degrees at which the arc should begin.
- `type`—How the arc should be closed. `ROUND` indicates that the starting and ending points of the arc should be connected to the center point by straight lines. You also may choose `OPEN`, which does not connect the start and end points, or `CHORD`, which connects the start and end points with a straight line.

22.3.2 CSS That Styles the Two-Dimensional Shapes

Figure 22.4 shows the CSS for the `BasicShapes` app. In this CSS file, we define two CSS rules with ID selectors (`#line1` and `#line2`) to style the app's two `Line` objects. The remaining rules use **type selectors**, which apply to all objects of a given type. You specify a type selector by using the JavaFX class name.

```
   1   /* Fig. 22.4: BasicShapes.css */
2   /* CSS that styles various two-dimensional shape
```

```
3
4   Line, Rectangle, Circle, Ellipse, Arc {
5       -fx-stroke-width: 10;
6   }
7
8   #line1 {
9       -fx-stroke: red;
10  }
11
12  #line2 {
13      -fx-stroke: rgba(0%, 50%, 0%, 0.5);
14      -fx-stroke-line-cap: round;
15  }
16
17  Rectangle {
18      -fx-stroke: red;
19      -fx-arc-width: 50;
20      -fx-arc-height: 50;
21      -fx-fill: yellow;
22  }
23
24  Circle {
25      -fx-stroke: blue;
26      -fx-fill: radial-gradient(center 50% 50%, rad
27  }
28
29  Ellipse {
30      -fx-stroke: green;
31      -fx-fill: image-pattern("yellowflowers.png");
32  }
33
34  Arc {
35      -fx-stroke: purple;
36      -fx-fill: linear-gradient(to right, cyan, whi
37  }
```

Fig. 22.4

CSS that styles various two-dimensional shapes.

Specifying Common Attributes for Various Objects

The CSS rule in lines 4–6 defines the `-fx-stroke-width` CSS property for all the shapes in the app—this property specifies the thickness of the `Line`s and the border thickness of all the other shapes. To apply this rule to multiple shapes we use CSS type selectors in a comma-separated list. So, line 4 indicates that the rule in lines 4–6 should be applied to all objects of types `Line`, `Rectangle`, `Circle`, `Ellipse` and `Arc` in the GUI.

Styling the `Line`s

The CSS rule in lines 8–10 sets the `-fx-stroke` to the solid color `red`. This rule applies to the `Line` with the `fx:id "line1"`. This rule is in addition to the rule at lines 4–6, which sets the stroke width for all `Line`s (and all the other shapes). When JavaFX renders an object, it combines all the CSS rules that apply to the object to determine its appearance. This rule applies to the `Line` with the `fx:id "line1"`.

Colors may be specified as

- named colors (such as `"red"`, `"green"` and `"blue"`),
- colors defined by their red, green, blue and alpha (transparency) components,
- colors defined by their hue, saturation, brightness and alpha components,

and more. For details on all the ways to specify color in CSS, see

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

The CSS rule in lines 12–15 applies to the `Line` with the `fx:id "line2"`. For this rule, we specified the `-fx-stroke` property's color using the CSS function `rgba`, which defines a color based on its red, green, blue and alpha (transparency) components. Here we used the version of `rgba` that receives percentages from 0% to 100% specifying the amount of red, green and blue in the color, and a value from 0.0 (transparent) to 1.0 (opaque) for the alpha component. Line 13 produces a semitransparent green line. You can see the interaction between the two `Line`s' colors at the intersection point in Fig. 22.3's output windows. The `-fx-stroke-line-cap` CSS property (line 14) indicates that the ends of the `Line` should be *rounded*—the rounding effect becomes more noticeable with thicker strokes.

Styling the `Rectangle`

For `Rectangle`s, `Circle`s, `Ellipse`s and `Arc`s you can specify both the `-fx-stroke` for the shapes' borders and the `-fx-fill`, which specifies the color or pattern that appears inside the shape. The rule in lines 17–22 uses a CSS type selector to indicate that all `Rectangle`s should have `red` borders (line 18) and `yellow` fill (line 21). Lines 19–20 define the `Rectangle`'s `-fx-arc-width` and `-fx-arc-height` properties, which specify the width and height of an ellipse that's divided in half horizontally and vertically, then used to round the `Rectangle`'s corners. Because these properties have the same value (`50`) in this app, the four corners are each one quarter of a circle with a diameter of 50.

Styling the `Circle`

The CSS rule at lines 24–27 applies to all `Circle` objects. Line 25 sets the `Circle`'s stroke to `blue`. Line 26 sets the `Circle`'s fill with a **gradient**—colors that transition gradually from one color to the next. You can transition between as many colors as you like and specify the points at which to change colors, called **color stops**. You can use gradients for any property that specifies a color. In this case, we use the CSS function `radial-gradient` in which the color changes gradually from a center point outward. The fill

```
   -fx-fill: radial-gradient(center 50% 50%, radius 60%,
```

indicates that the gradient should begin from a `center` point

at `50% 50%`—the middle of the shape horizontally and the middle of the shape vertically. The `radius` specifies the distance from the center at which an even mixture of the two colors appears. This radial gradient begins with the color `white` in the center and ends with `red` at the outer edge of the `Circle`'s fill. We'll discuss a linear gradient momentarily.

Styling the `Ellipse`

The CSS rule at lines 29–32 applies to all `Ellipse` objects. Line 30 specifies that an `Ellipse` should have a `green` stroke. Line 31 specifies that the `Ellipse`'s fill should be the image in the file `yellowflowers.png`, which is located in this app's folder. This image is provided in the `images` folder with the chapter's examples—if you're building this app from scratch, copy the video into the app's folder on your system. To specify an image as fill, you use the CSS function `image-pattern`. [*Note:* At the time of this writing, Scene Builder does not display a shape's fill correctly if it's specified with a CSS `image-pattern`. You must run the example to see the fill, as shown in Fig. 22.3(b).]

Styling the `Arc`

The CSS rule at lines 34–37 applies to all `Arc` objects. Line 35 specifies that an `Arc` should have a `purple` stroke. In this

case, line 36

```
   -fx-fill: linear-gradient(to right, cyan, white);
```

specifies that the `Arc` should be filled with a **linear gradient**—such gradients gradually transition from one color to the next horizontally, vertically or diagonally. You can transition between as many colors as you like and specify the points at which to change colors. To create a linear gradient, you use the CSS function `linear-gradient`. In this case, `to right` indicates that the gradient should start from the shape's left edge and transition through colors to the shape's right edge. We specified only two colors here—`cyan` at the left edge of the gradient and `white` at the right edge—but two or more colors can be specified in the comma-separated list. For more information on all the options for configuring radial gradients, linear gradients and image patterns, see

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