Chapter 22
JavaFX Graphics and Multimedia

Java How to Program, 11/e, Global Edition

Questions? E-mail paul.deitel@deitel.com

OBJECTIVES

In this chapter you'll:

- Use JavaFX graphics and multimedia capabilities to make your apps "come alive" with graphics, animations, audio and video.
- Use external Cascading Style Sheets to customize the look of **Node**s while maintaining their functionality.

OBJECTIVES (cont.)

- Customize fonts attributes such as font family, size and style.
- Display two-dimensional shape nodes of types `Line`, `Rectangle`, `Circle`, `Ellipse`, `Arc`, `Path`, `Polyline` and `Polygon`.
- Customize the stroke and fill of shapes with solid colors, images and gradients.
- Use `Transforms` to reposition and reorient nodes.

OBJECTIVES (cont.)

- Display and control video playback with `Media`, `MediaPlayer` and `MediaView`.
- Animate `Node` properties with `Transition` and `Timeline` animations.
- Use an `AnimationTimer` to create frame-by-frame animations.
- Draw graphics on a `Canvas` node.
- Display 3D shapes.

22.3 Displaying Two-Dimensional Shapes

- 22.3.1 Defining Two-Dimensional Shapes with FXML
- 22.3.2 CSS That Styles the Two-Dimensional Shapes

22.4 `Polylines`, `Polygons` and `Paths`

- 22.4.1 GUI and CSS
- 22.4.2 `PolyShapesController` Class

22.5 Transforms

22.6 Playing Video with `Media`, `MediaPlayer` and `MediaViewer`

22.6.1 `VideoPlayer` GUI
22.6.2 `VideoPlayerController` Class

22.7 `Transition` Animations

22.7.1 `TransitionAnimations.fxml`
22.7.2 `TransitionAnimations-Controller` Class

```
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 }
```

Fig. 22.1 | CSS rules that style the VBox and Labels. (Part 1 of 2.)

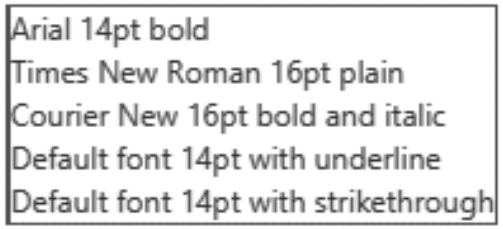
```
20
21   #label4 {
22      -fx-font-size: 14pt;
23      -fx-underline: true;
24   }
25
26   #label5 {
27      -fx-font-size: 14pt;
28   }
29
30   #label5 .text {
31      -fx-strikethrough: true;
32   }
```

Fig. 22.1 | CSS rules that style the VBox and Labels. (Part 2 of 2.)

```
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 bold" />
13       <Label fx:id="label2" text="Times New Roman 16pt plain" />
14       <Label fx:id="label3" text="Courier New 16pt bold and italic" />
15       <Label fx:id="label4" text="Default font 14pt with underline" />
16       <Label fx:id="label5" text="Default font 14pt with strikethrough" />
17    </children>
18 </VBox>
```

Fig. 22.2 | FontCSS GUI that is styled via external CSS. (Part 1 of 3.)

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

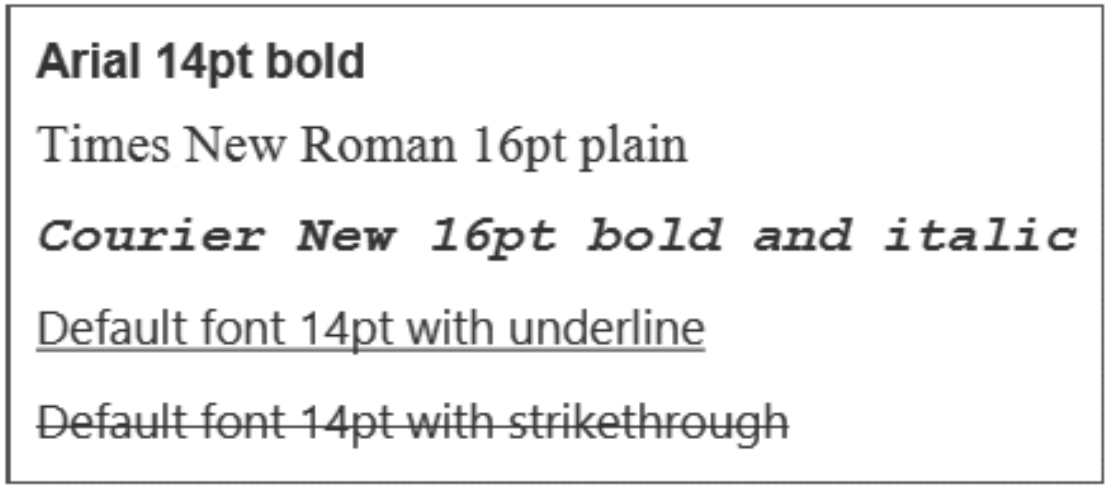


Fig. 22.2 | **FontCSS** GUI that is styled via external CSS. (Part 2 of 3.)

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

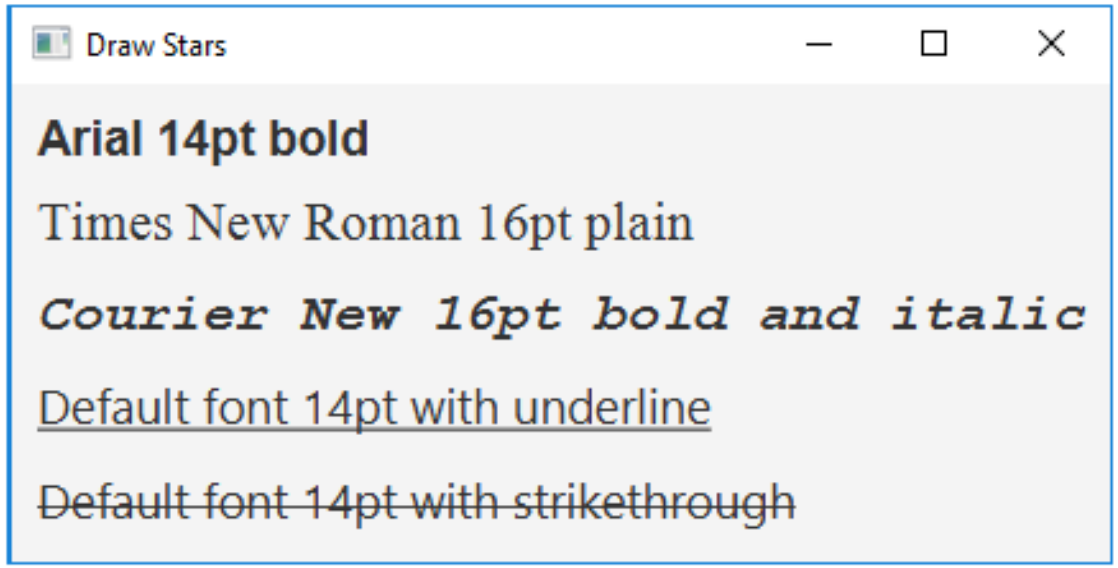


Fig. 22.2 | **FontCSS** GUI that is styled via external CSS. (Part 3 of 3.)

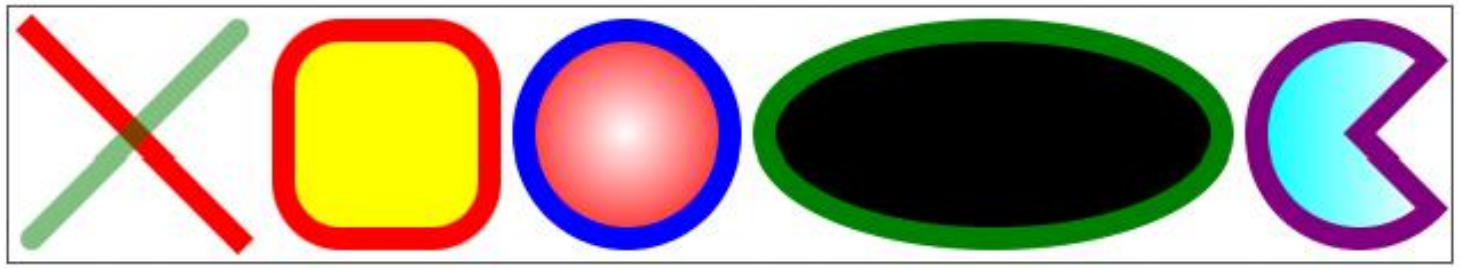
```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- Fig. 22.3: BasicShapes.fxml -->
3  <!-- Defining Shape objects and styling via CSS -->
4
5  <?import javafx.scene.layout.Pane?>
6  <?import javafx.scene.shape.Arc?>
7  <?import javafx.scene.shape.Circle?>
8  <?import javafx.scene.shape.Ellipse?>
9  <?import javafx.scene.shape.Line?>
10 <?import javafx.scene.shape.Rectangle?>
11
12 <Pane id="Pane" prefHeight="110.0" prefWidth="630.0"
13    stylesheets="@BasicShapes.css" xmlns="http://javafx.com/javafx/8.0.60"
14    xmlns:fx="http://javafx.com/fxml/1">
```

Fig. 22.3 | Defining Shape objects and styling via CSS. (Part 1 of 3.)

```
15      <children>
16         <Line fx:id="line1" endX="100.0" endY="100.0"
17            startX="10.0" startY="10.0" />
18         <Line fx:id="line2" endX="10.0" endY="100.0"
19            startX="100.0" startY="10.0" />
20         <Rectangle fx:id="rectangle" height="90.0" layoutX="120.0"
21            layoutY="10.0" width="90.0" />
22         <Circle fx:id="circle" centerX="270.0" centerY="55.0"
23            radius="45.0" />
24         <Ellipse fx:id="ellipse" centerX="430.0" centerY="55.0"
25            radiusX="100.0" radiusY="45.0" />
26         <Arc fx:id="arc" centerX="590.0" centerY="55.0" length="270.0"
27            radiusX="45.0" radiusY="45.0" startAngle="45.0" type="ROUND" />
28      </children>
29   </Pane>
```

Fig. 22.3 | Defining Shape objects and styling via CSS. (Part 2 of 3.)

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.

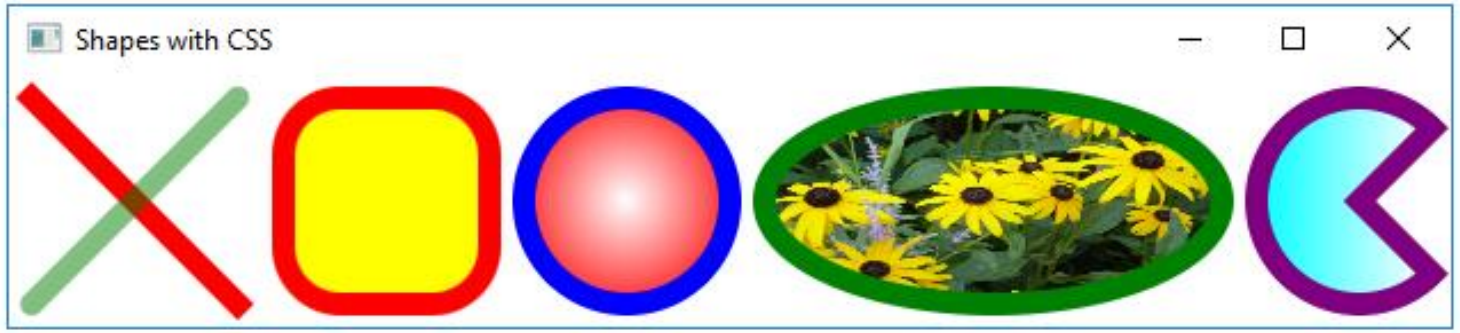


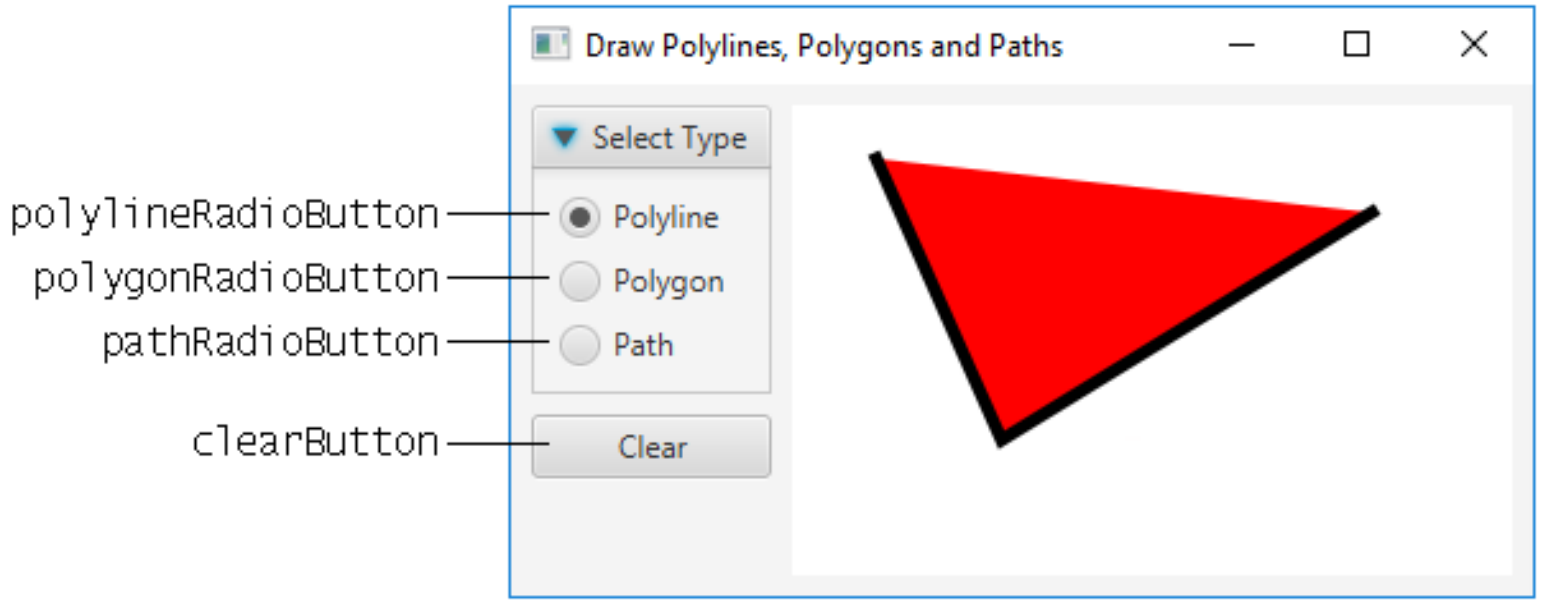
Fig. 22.3 | Defining **Shape** objects and styling via CSS. (Part 3 of 3.)

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

Fig. 22.4 | CSS that styles various two-dimensional shapes. (Part 1 of 2.)

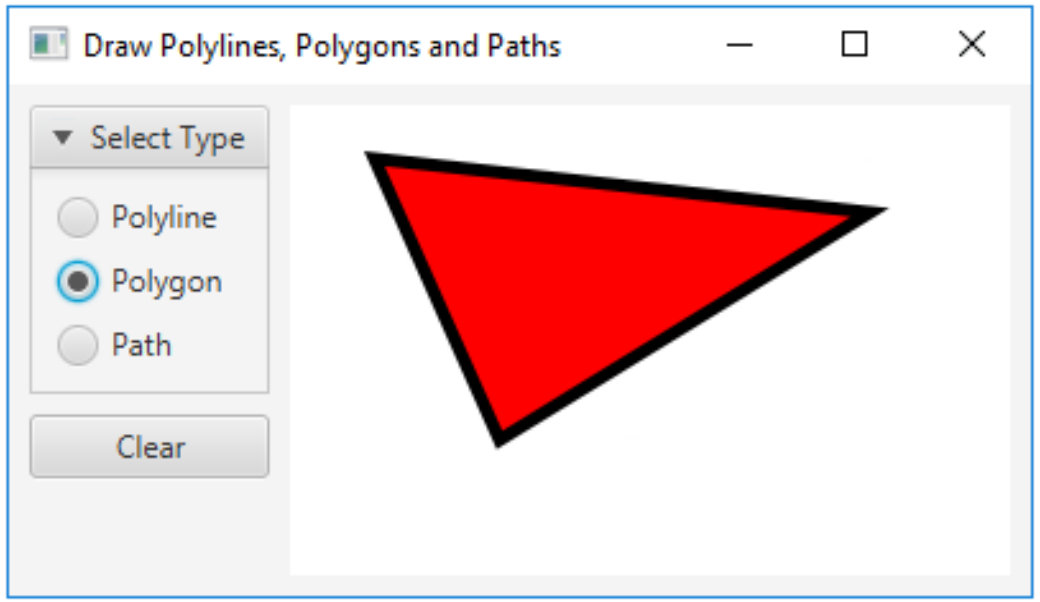
```
17  Rectangle {
18     -fx-stroke: red;
19     -fx-arc-width: 50;
20     -fx-arc-height: 50;
21     -fx-fill: yellow;
22  }
23
24  Circle {
25     -fx-stroke: blue;
26     -fx-fill: radial-gradient(center 50% 50%, radius 60%, white, red);
27  }
28
29  Ellipse {
30     -fx-stroke: green;
31     -fx-fill: image-pattern("yellowflowers.png");
32  }
33
34  Arc {
35     -fx-stroke: purple;
36     -fx-fill: linear-gradient(to right, cyan, white);
37  }
```

Fig. 22.4 | CSS that styles various two-dimensional shapes. (Part 2 of 2.)



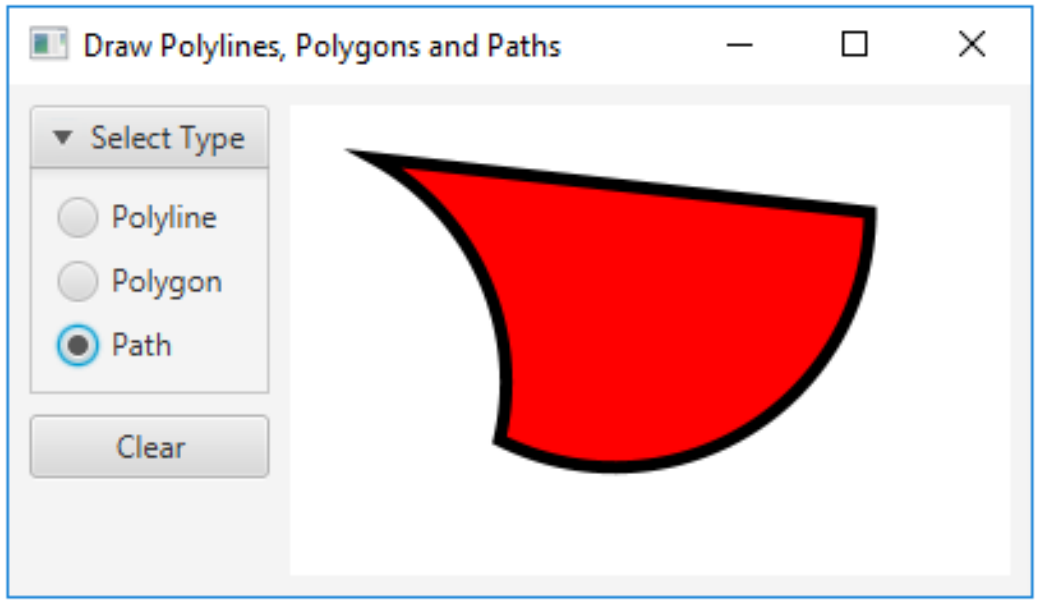
a) `Polyline` with three points—the starting and ending points are not connected

Fig. 22.5 | **`Polyline`s, `Polygon`s and `Path`s.** (Part 1 of 3.)



b) `Polygon` with three points—the starting and ending points are connected automatically

Fig. 22.5 | `Polylines`, `Polygons` and `Paths`. (Part 2 of 3.)



c) `Path` with two arc segments and a line connecting the first and last points

Fig. 22.5 | `Polylines`, `Polygons` and `Paths`. (Part 3 of 3.)

```
1  // Fig. 22.6: PolyShapesController.java
2  // Drawing Polylines, Polygons and Paths.
3  import javafx.event.ActionEvent;
4  import javafx.fxml.FXML;
5  import javafx.scene.control.RadioButton;
6  import javafx.scene.control.ToggleGroup;
7  import javafx.scene.input.MouseEvent;
8  import javafx.scene.shape.ArcTo;
9  import javafx.scene.shape.ClosePath;
10 import javafx.scene.shape.MoveTo;
11 import javafx.scene.shape.Path;
12 import javafx.scene.shape.Polygon;
13 import javafx.scene.shape.Polyline;
14
```

Fig. 22.6 | Drawing `Polylines`, `Polygons` and `Paths`. (Part 1 of 5.)

```
15  public class PolyShapesController {
16     // enum representing shape types
17     private enum ShapeType {POLYLINE, POLYGON, PATH};
18
19     // instance variables that refer to GUI components
20     @FXML private RadioButton polylineRadioButton;
21     @FXML private RadioButton polygonRadioButton;
22     @FXML private RadioButton pathRadioButton;
23     @FXML private ToggleGroup toggleGroup;
24     @FXML private Polyline polyline;
25     @FXML private Polygon polygon;
26     @FXML private Path path;
27
28     // instance variables for managing state
29     private ShapeType shapeType = ShapeType.POLYLINE;
30     private boolean sweepFlag = true; // used with arcs in a Path
31
```

Fig. 22.6 | Drawing `Polylines`, `Polygons` and `Paths`. (Part 2 of 5.)

```
32      // set user data for the RadioButtons and display polyline object
33      public void initialize() {
34         // user data on a control can be any Object
35         polylineRadioButton.setUserData(ShapeType.POLYLINE);
36         polygonRadioButton.setUserData(ShapeType.POLYGON);
37         pathRadioButton.setUserData(ShapeType.PATH);
38
39         displayShape(); // sets polyline's visibility to true when app loads
40      }
41
```

Fig. 22.6 | Drawing `Polylines`, `Polygons` and `Paths`. (Part 3 of 5.)

```
42      // handles drawingArea's onMouseClicked event
43      @FXML
44      private void drawingAreaMouseClicked(MouseEvent e) {
45         polyline.getPoints().addAll(e.getX(), e.getY());
46         polygon.getPoints().addAll(e.getX(), e.getY());
47
48         // if path is empty, move to first click position and close path
49         if (path.getElements().isEmpty()) {
50            path.getElements().add(new MoveTo(e.getX(), e.getY()));
51            path.getElements().add(new ClosePath());
52         }
53         else { // insert a new path segment before the ClosePath element
54            // create an arc segment and insert it in the path
55            ArcTo arcTo = new ArcTo();
56            arcTo.setX(e.getX());
57            arcTo.setY(e.getY());
58            arcTo.setRadiusX(100.0);
59            arcTo.setRadiusY(100.0);
60            arcTo.setSweepFlag(sweepFlag);
61            sweepFlag = !sweepFlag;
62            path.getElements().add(path.getElements().size() - 1, arcTo);
63         }
64      }
65
```

Fig. 22.6 | Drawing `Polylines`, `Polygons` and `Paths`. (Part 4 of 5.)

```
66      // handles color RadioButton's ActionEvents
67      @FXML
68      private void shapeRadioButtonSelected(ActionEvent e) {
69         // user data for each color RadioButton is a ShapeType constant
70         shapeType =
71            (ShapeType) toggleGroup.getSelectedToggle().getUserData();
72         displayShape(); // display the currently selected shape
73      }
74
75      // displays currently selected shape
76      private void displayShape() {
77         polyline.setVisible(shapeType == ShapeType.POLYLINE);
78         polygon.setVisible(shapeType == ShapeType.POLYGON);
79         path.setVisible(shapeType == ShapeType.PATH);
80      }
81
82      // resets each shape
83      @FXML
84      private void clearButtonPressed(ActionEvent event) {
85         polyline.getPoints().clear();
86         polygon.getPoints().clear();
87         path.getElements().clear();
88      }
89   }
```

Fig. 22.6 | Drawing `Polylines`, `Polygons` and `Paths`. (Part 5 of 5.)

```
1  // Fig. 22.7: DrawStarsController.java
2  // Create a circle of stars using Polygons and Rotate transforms
3  import java.security.SecureRandom;
4  import javafx.fxml.FXML;
5  import javafx.scene.layout.Pane;
6  import javafx.scene.paint.Color;
7  import javafx.scene.shape.Polygon;
8  import javafx.scene.transform.Transform;
9
10 public class DrawStarsController {
11    @FXML private Pane pane;
12    private static final SecureRandom random = new SecureRandom();
13
14    public void initialize() {
15       // points that define a five-pointed star shape
16       Double[] points = {205.0,150.0, 217.0,186.0, 259.0,186.0,
17          223.0,204.0, 233.0,246.0, 205.0,222.0, 177.0,246.0, 187.0,204.0,
18          151.0,186.0, 193.0,186.0};
19
```

Fig. 22.7 | Create a circle of stars using `Polygons` and `Rotate` transforms. (Part 1 of 3.)

```
20          // create 18 stars
21          for (int count = 0; count < 18; ++count) {
22             // create a new Polygon and copy existing points into it
23             Polygon newStar = new Polygon();
24             newStar.getPoints().addAll(points);
25
26             // create random Color and set as newStar's fill
27             newStar.setStroke(Color.GREY);
28             newStar.setFill(Color.rgb(random.nextInt(255),
29                random.nextInt(255), random.nextInt(255),
30                random.nextDouble()));
31
32             // apply a rotation to the shape
33             newStar.getTransforms().add(
34                Transform.rotate(count * 20, 150, 150));
35             pane.getChildren().add(newStar);
36          }
37       }
38    }
```

Fig. 22.7 | Create a circle of stars using `Polygons` and `Rotate` transforms. (Part 2 of 3.)

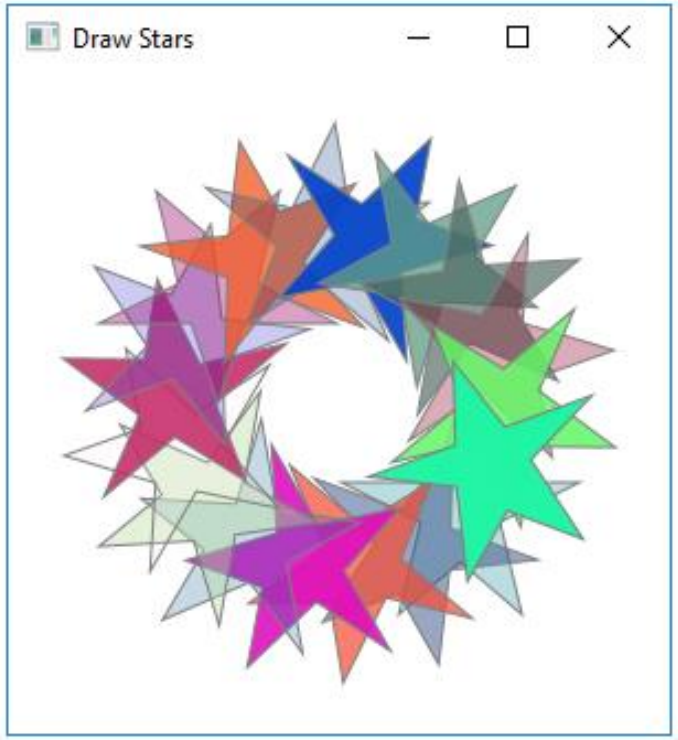


Fig. 22.7 | Create a circle of stars using `Polygon`s and `Rotate` transforms. (Part 3 of 3.)

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- Fig. 22.8: VideoPlayer.fxml -->
3  <!-- VideoPlayer GUI with a MediaView and a Button -->
4
5  <?import javafx.scene.control.Button?>
6  <?import javafx.scene.control.ToolBar?>
7  <?import javafx.scene.layout.BorderPane?>
8  <?import javafx.scene.media.MediaView?>
9
10 <BorderPane prefHeight="400.0" prefWidth="600.0"
11    style="-fx-background-color: black;"
12    xmlns="http://javafx.com/javafx/8.0.60"
13    xmlns:fx="http://javafx.com/fxml/1"
14    fx:controller="VideoPlayerController">
```

Fig. 22.8 | `VideoPlayer` GUI with a `MediaView` and a `Button`. Video courtesy of NASA—see `http://www.nasa.gov/multimedia/guidelines/` for usage guidelines. (Part 1 of 4.)

```
15      <bottom>
16         <ToolBar prefHeight="40.0" prefWidth="200.0"
17            BorderPane.alignment="CENTER">
18            <items>
19               <Button fx:id="playPauseButton"
20                  onAction="#playPauseButtonPressed" prefHeight="25.0"
21                  prefWidth="60.0" text="Play" />
22            </items>
23         </ToolBar>
24      </bottom>
25      <center>
26         <MediaView fx:id="mediaView" BorderPane.alignment="CENTER" />
27      </center>
28   </BorderPane>
```

Fig. 22.8 | `VideoPlayer` GUI with a `MediaView` and a `Button`. Video courtesy of NASA—see `http://www.nasa.gov/multimedia/guidelines/` for usage guidelines. (Part 2 of 4.)

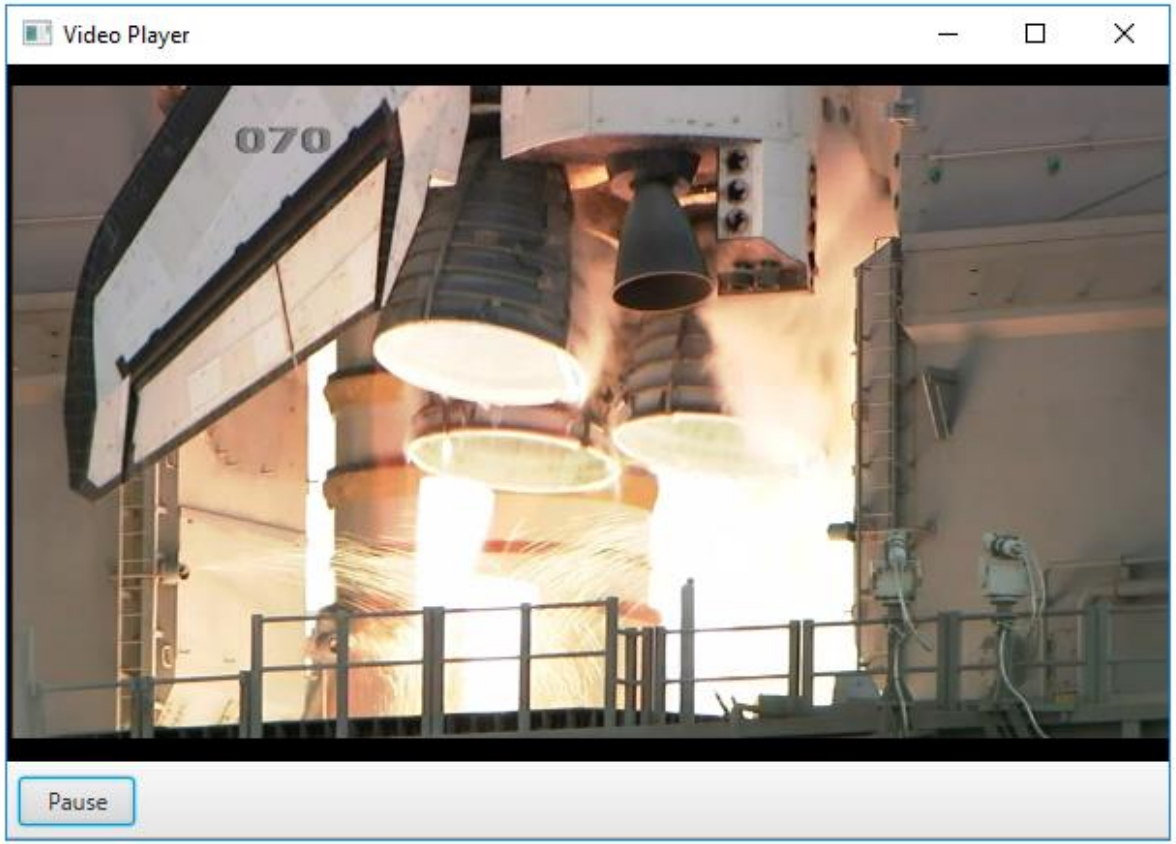


Fig. 22.8 | `VideoPlayer` GUI with a `MediaView` and a `Button`. Video courtesy of NASA—see `http://www.nasa.gov/multimedia/guidelines/` for usage guidelines. (Part 3 of 4.)

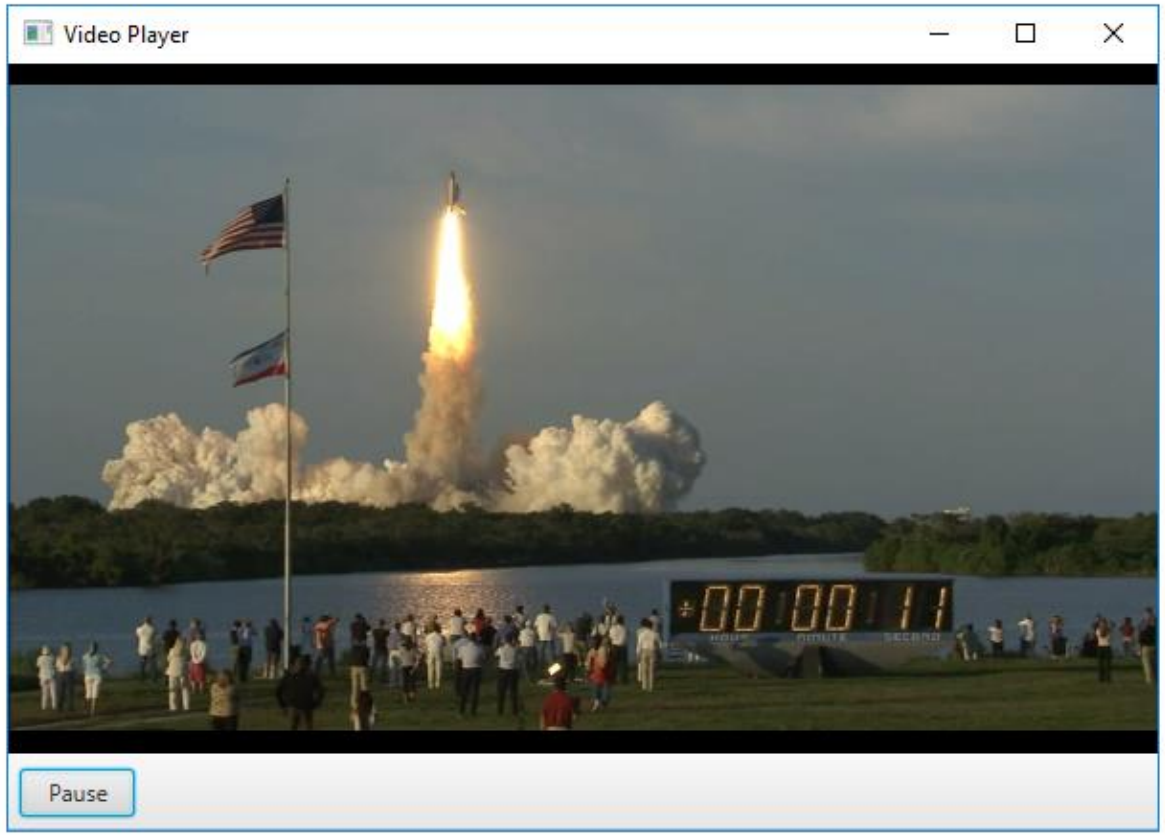


Fig. 22.8 | `VideoPlayer` GUI with a `MediaView` and a `Button`. Video courtesy of NASA—see `http://www.nasa.gov/multimedia/guidelines/` for usage guidelines. (Part 4 of 4.)

```
1  // Fig. 22.9: VideoPlayerController.java
2  // Using Media, MediaPlayer and MediaView to play a video.
3  import java.net.URL;
4  import javafx.beans.binding.Bindings;
5  import javafx.beans.property.DoubleProperty;
6  import javafx.event.ActionEvent;
7  import javafx.fxml.FXML;
8  import javafx.scene.control.Button;
9  import javafx.scene.media.Media;
10 import javafx.scene.media.MediaPlayer;
11 import javafx.scene.media.MediaView;
12 import javafx.util.Duration;
13 import org.controlsfx.dialog.ExceptionDialog;
14
```

Fig. 22.9 | Using `Media`, `MediaPlayer` and `MediaView` to play a video. (Part 1 of 6.)

```
15  public class VideoPlayerController {
16     @FXML private MediaView mediaView;
17     @FXML private Button playPauseButton;
18     private MediaPlayer mediaPlayer;
19     private boolean playing = false;
20
21     public void initialize() {
22        // get URL of the video file
23        URL url = VideoPlayerController.class.getResource("sts117.mp4");
24
25        // create a Media object for the specified URL
26        Media media = new Media(url.toExternalForm());
27
28        // create a MediaPlayer to control Media playback
29        mediaPlayer = new MediaPlayer(media);
30
31        // specify which MediaPlayer to display in the MediaView
32        mediaView.setMediaPlayer(mediaPlayer);
```

Fig. 22.9 | Using `Media`, `MediaPlayer` and `MediaView` to play a video. (Part 2 of 6.)

```
33
34          // set handler to be called when the video completes playing
35          mediaPlayer.setOnEndOfMedia(
36             new Runnable() {
37                public void run() {
38                   playing = false;
39                   playPauseButton.setText("Play");
40                   mediaPlayer.seek(Duration.ZERO);
41                   mediaPlayer.pause();
42                }
43             }
44          );
45
```

Fig. 22.9 | Using `Media`, `MediaPlayer` and `MediaView` to play a video. (Part 3 of 6.)

```
46          // set handler that displays an ExceptionDialog if an error occurs
47          mediaPlayer.setOnError(
48             new Runnable() {
49                public void run() {
50                   ExceptionDialog dialog =
51                      new ExceptionDialog(mediaPlayer.getError());
52                   dialog.showAndWait();
53                }
54             }
55          );
56
```

Fig. 22.9 | Using `Media`, `MediaPlayer` and `MediaView` to play a video. (Part 4 of 6.)

```
57          // set handler that resizes window to video size once ready to play
58          mediaPlayer.setOnReady(
59             new Runnable() {
60                public void run() {
61                   DoubleProperty width = mediaView.fitWidthProperty();
62                   DoubleProperty height = mediaView.fitHeightProperty();
63                   width.bind(Bindings.selectDouble(
64                      mediaView.sceneProperty(), "width"));
65                   height.bind(Bindings.selectDouble(
66                      mediaView.sceneProperty(), "height"));
67                }
68             }
69          );
70       }
71
```

Fig. 22.9 | Using `Media`, `MediaPlayer` and `MediaView` to play a video. (Part 5 of 6.)

```
72        // toggle media playback and the text on the playPauseButton
73        @FXML
74        private void playPauseButtonPressed(ActionEvent e) {
75           playing = !playing;
76
77           if (playing) {
78              playPauseButton.setText("Pause");
79              mediaPlayer.play();
80           }
81           else {
82              playPauseButton.setText("Play");
83              mediaPlayer.pause();
84           }
85        }
86     }
```

Fig. 22.9 | Using `Media`, `MediaPlayer` and `MediaView` to play a video. (Part 6 of 6.)

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- Fig. 22.10: TransitionAnimations.fxml -->
3  <!-- FXML for a Rectangle and Button -->
4
5  <?import javafx.scene.control.Button?>
6  <?import javafx.scene.layout.Pane?>
7  <?import javafx.scene.shape.Rectangle?>
8
9  <Pane id="Pane" prefHeight="200.0" prefWidth="180.0"
10    stylesheets="@TransitionAnimations.css"
11    xmlns="http://javafx.com/javafx/8.0.60"
12    xmlns:fx="http://javafx.com/fxml/1"
13    fx:controller="TransitionAnimationsController">
14    <children>
15       <Rectangle fx:id="rectangle" height="90.0" layoutX="45.0"
16          layoutY="45.0" width="90.0" />
17       <Button fx:id="startButton" layoutX="38.0" layoutY="161.0"
18          mnemonicParsing="false"
19          onAction="#startButtonPressed" text="Start Animations" />
20    </children>
21 </Pane>
```

Fig. 22.10 | FXML for a `Rectangle` and `Button`. (Part 1 of 7.)

a) Initial Rectangle

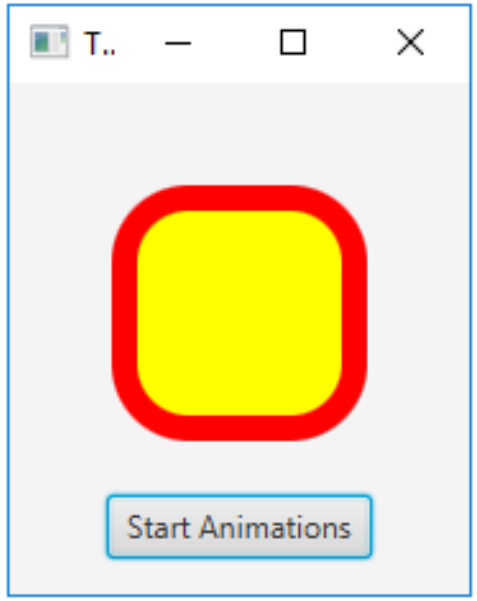


Fig. 22.10 | FXML for a **`Rectangle`** and **`Button`**. (Part 2 of 7.)

b) Rectangle undergoing parallel fill and stroke transitions

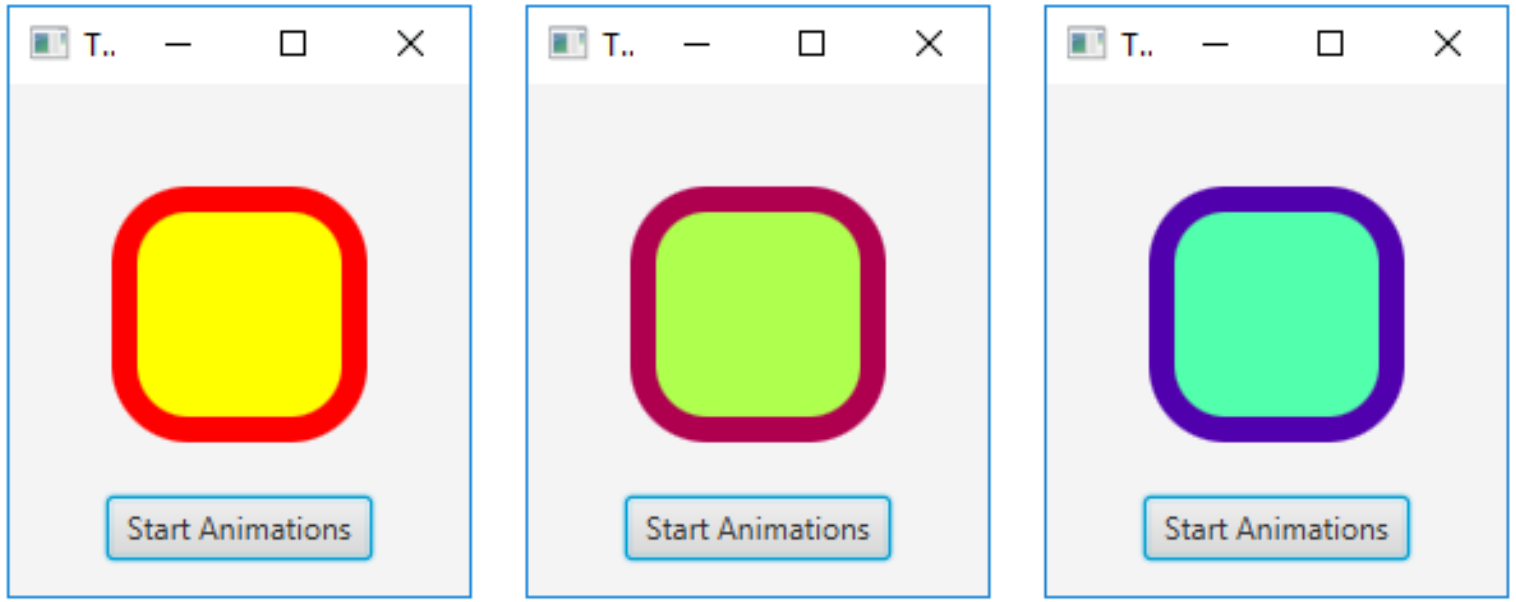


Fig. 22.10 | FXML for a **Rectangle** and **Button**. (Part 3 of 7.)

c) Rectangle undergoing a fade transition

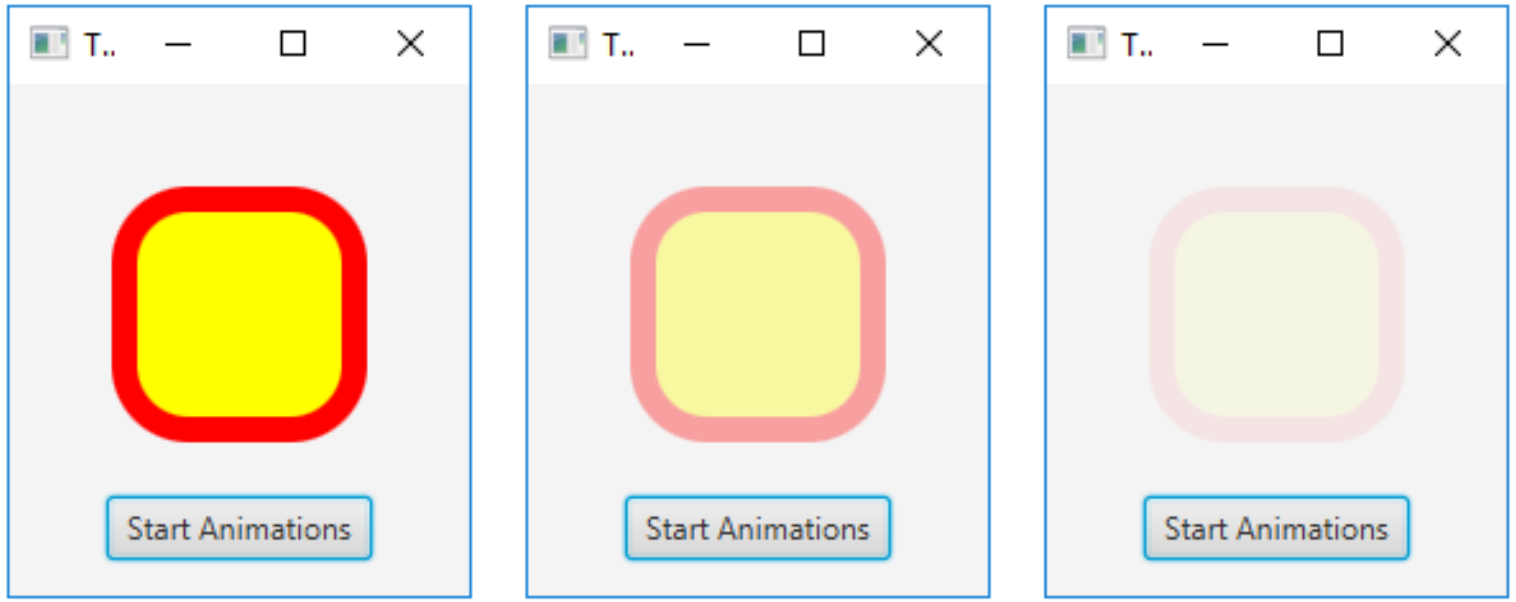


Fig. 22.10 | FXML for a **Rectangle** and **Button**. (Part 4 of 7.)

d) Rectangle undergoing a rotate transition

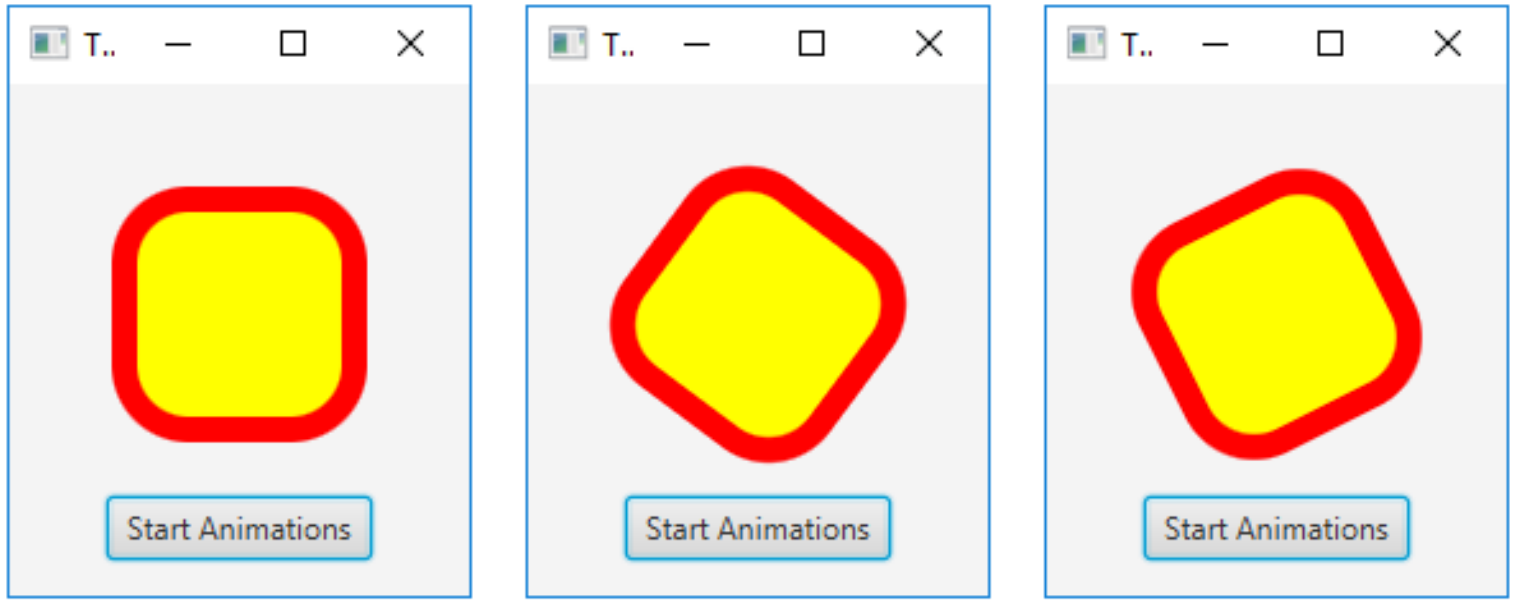


Fig. 22.10 | FXML for a **Rectangle** and **Button**. (Part 5 of 7.)

e) Rectangle undergoing a path transition

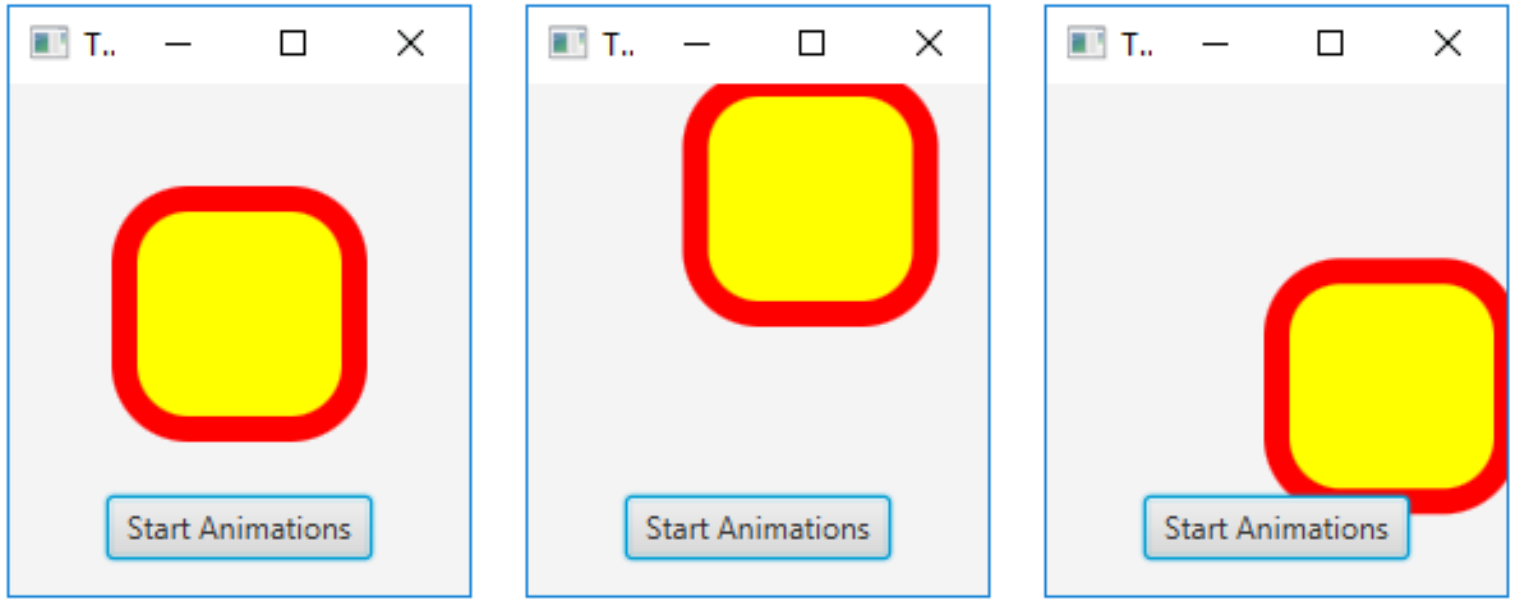


Fig. 22.10 | FXML for a **Rectangle** and **Button**. (Part 6 of 7.)

f) Rectangle undergoing a scale transition

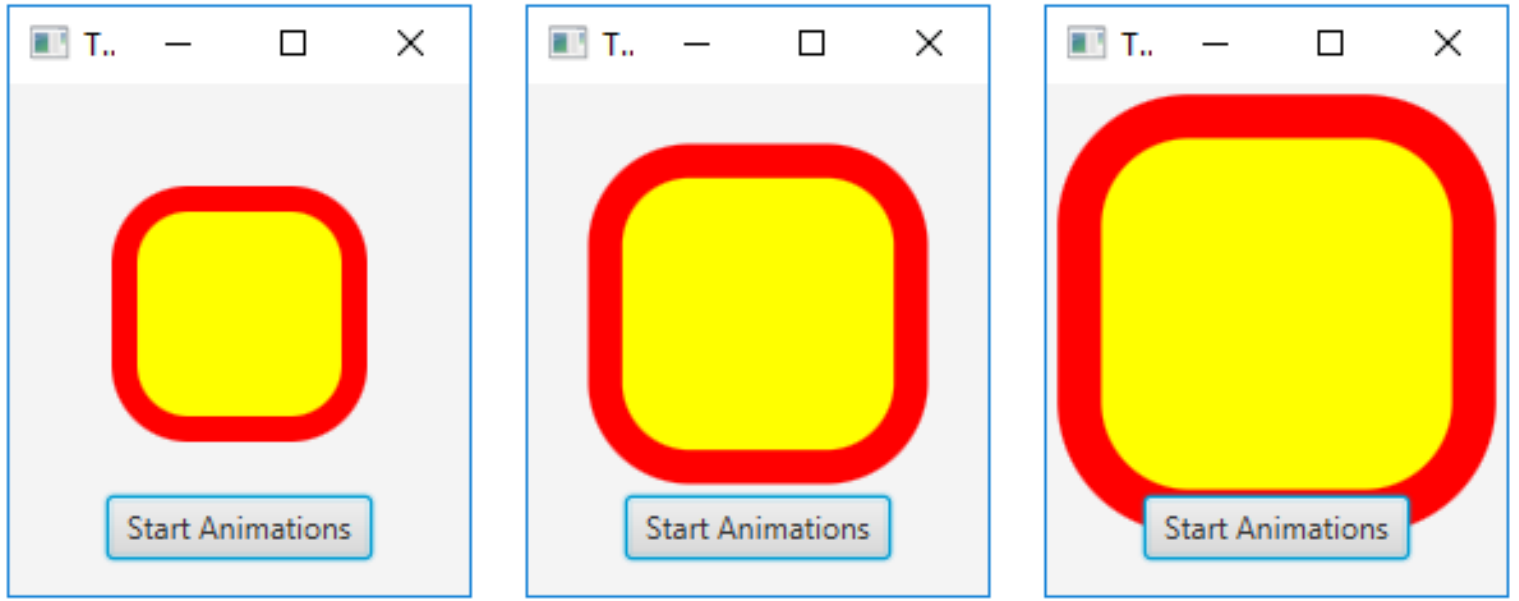


Fig. 22.10 | FXML for a **Rectangle** and **Button**. (Part 7 of 7.)

```
1  // Fig. 22.11: TransitionAnimationsController.java
2  // Applying Transition animations to a Rectangle.
3  import javafx.animation.FadeTransition;
4  import javafx.animation.FillTransition;
5  import javafx.animation.Interpolator;
6  import javafx.animation.ParallelTransition;
7  import javafx.animation.PathTransition;
8  import javafx.animation.RotateTransition;
9  import javafx.animation.ScaleTransition;
10 import javafx.animation.SequentialTransition;
11 import javafx.animation.StrokeTransition;
12 import javafx.event.ActionEvent;
13 import javafx.fxml.FXML;
14 import javafx.scene.paint.Color;
15 import javafx.scene.shape.LineTo;
16 import javafx.scene.shape.MoveTo;
17 import javafx.scene.shape.Path;
18 import javafx.scene.shape.Rectangle;
19 import javafx.util.Duration;
```

Fig. 22.11 | Applying `Transition` animations to a `Rectangle`. (Part 1 of 5.)

```
20
21   public class TransitionAnimationsController {
22      @FXML private Rectangle rectangle;
23
24      // configure and start transition animations
25      @FXML
26      private void startButtonPressed(ActionEvent event) {
27         // transition that changes a shape's fill
28         FillTransition fillTransition =
29            new FillTransition(Duration.seconds(1));
30         fillTransition.setToValue(Color.CYAN);
31         fillTransition.setCycleCount(2);
32
33         // each even cycle plays transition in reverse to restore original
34         fillTransition.setAutoReverse(true);
35
```

Fig. 22.11 | Applying `Transition` animations to a `Rectangle`. (Part 2 of 5.)

```
36          // transition that changes a shape's stroke over time
37          StrokeTransition strokeTransition =
38             new StrokeTransition(Duration.seconds(1));
39          strokeTransition.setToValue(Color.BLUE);
40          strokeTransition.setCycleCount(2);
41          strokeTransition.setAutoReverse(true);
42
43          // parallelizes multiple transitions
44          ParallelTransition parallelTransition =
45             new ParallelTransition(fillTransition, strokeTransition);
46
47          // transition that changes a node's opacity over time
48          FadeTransition fadeTransition =
49             new FadeTransition(Duration.seconds(1));
50          fadeTransition.setFromValue(1.0); // opaque
51          fadeTransition.setToValue(0.0); // transparent
52          fadeTransition.setCycleCount(2);
53          fadeTransition.setAutoReverse(true);
```

Fig. 22.11 | Applying `Transition` animations to a `Rectangle`. (Part 3 of 5.)

```
54
55          // transition that rotates a node
56          RotateTransition rotateTransition =
57             new RotateTransition(Duration.seconds(1));
58          rotateTransition.setByAngle(360.0);
59          rotateTransition.setCycleCount(2);
60          rotateTransition.setInterpolator(Interpolator.EASE_BOTH);
61          rotateTransition.setAutoReverse(true);
62
63          // transition that moves a node along a Path
64          Path path = new Path(new MoveTo(45, 45), new LineTo(45, 0),
65             new LineTo(90, 0), new LineTo(90, 90), new LineTo(0, 90));
66          PathTransition translateTransition =
67             new PathTransition(Duration.seconds(2), path);
68          translateTransition.setCycleCount(2);
69          translateTransition.setInterpolator(Interpolator.EASE_IN);
70          translateTransition.setAutoReverse(true);
71
```

Fig. 22.11 | Applying `Transition` animations to a `Rectangle`. (Part 4 of 5.)

```
72          // transition that scales a shape to make it larger or smaller
73          ScaleTransition scaleTransition =
74             new ScaleTransition(Duration.seconds(1));
75          scaleTransition.setByX(0.75);
76          scaleTransition.setByY(0.75);
77          scaleTransition.setCycleCount(2);
78          scaleTransition.setInterpolator(Interpolator.EASE_OUT);
79          scaleTransition.setAutoReverse(true);
80
81          // transition that applies a sequence of transitions to a node
82          SequentialTransition sequentialTransition =
83             new SequentialTransition (rectangle, parallelTransition,
84                fadeTransition, rotateTransition, translateTransition,
85                scaleTransition);
86          sequentialTransition.play(); // play the transition
87       }
88    }
```

Fig. 22.11 | Applying `Transition` animations to a `Rectangle`. (Part 5 of 5.)

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- Fig. 22.12: TimelineAnimation.fxml -->
3  <!-- FXML for a Circle that will be animated by the controller -->
4
5  <?import javafx.scene.layout.Pane?>
6  <?import javafx.scene.shape.Circle?>
7
8  <Pane id="Pane" fx:id="pane" prefHeight="400.0"
9     prefWidth="600.0" xmlns:fx="http://javafx.com/fxml/1"
10    xmlns="http://javafx.com/javafx/8.0.60"
11    fx:controller="TimelineAnimationController">
12    <children>
13       <Circle fx:id="c" fill="DODGERBLUE" layoutX="142.0" layoutY="143.0"
14          radius="40.0" stroke="BLACK" strokeType="INSIDE"
15          strokeWidth="5.0" />
16    </children>
17 </Pane>
```

Fig. 22.12 | FXML for a `Circle` that will be animated by the controller.

```
1  // Fig. 22.13: TimelineAnimationController.java
2  // Bounce a circle around a window using a Timeline animation
3  import java.security.SecureRandom;
4  import javafx.animation.KeyFrame;
5  import javafx.animation.Timeline;
6  import javafx.event.ActionEvent;
7  import javafx.event.EventHandler;
8  import javafx.fxml.FXML;
9  import javafx.geometry.Bounds;
10 import javafx.scene.layout.Pane;
11 import javafx.scene.shape.Circle;
12 import javafx.util.Duration;
13
```

Fig. 22.13 | Bounce a circle around a window using a `Timeline` animation. (Part 1 of 6.)

```
14  public class TimelineAnimationController {
15     @FXML Circle c;
16     @FXML Pane pane;
17
18     public void initialize() {
19        SecureRandom random = new SecureRandom();
20
21        // define a timeline animation
22        Timeline timelineAnimation = new Timeline(
23           new KeyFrame(Duration.millis(10),
24              new EventHandler<ActionEvent>() {
25                 int dx = 1 + random.nextInt(5);
26                 int dy = 1 + random.nextInt(5);
27
```

Fig. 22.13 | Bounce a circle around a window using a `Timeline` animation. (Part 2 of 6.)

```
28                  // move the circle by the dx and dy amounts
29                  @Override
30                  public void handle(final ActionEvent e) {
31                     c.setLayoutX(c.getLayoutX() + dx);
32                     c.setLayoutY(c.getLayoutY() + dy);
33                     Bounds bounds = pane.getBoundsInLocal();
34
35                     if (hitRightOrLeftEdge(bounds)) {
36                        dx *= -1;
37                     }
38
39                     if (hitTopOrBottom(bounds)) {
40                        dy *= -1;
41                     }
42                  }
43               }
44            )
45         );
46
47         // indicate that the timeline animation should run indefinitely
48         timelineAnimation.setCycleCount(Timeline.INDEFINITE);
49         timelineAnimation.play();
50      }
```

Fig. 22.13 | Bounce a circle around a window using a `Timeline` animation. (Part 3 of 6.)

```
51
52      // determines whether the circle hit the left or right of the window
53      private boolean hitRightOrLeftEdge(Bounds bounds) {
54         return (c.getLayoutX() <= (bounds.getMinX() + c.getRadius())) ||
55            (c.getLayoutX() >= (bounds.getMaxX() - c.getRadius()));
56      }
57
58      // determines whether the circle hit the top or bottom of the window
59      private boolean hitTopOrBottom(Bounds bounds) {
60         return (c.getLayoutY() <= (bounds.getMinY() + c.getRadius())) ||
61            (c.getLayoutY() >= (bounds.getMaxY() - c.getRadius()));
62      }
63   }
```

Fig. 22.13 | Bounce a circle around a window using a `Timeline` animation. (Part 4 of 6.)

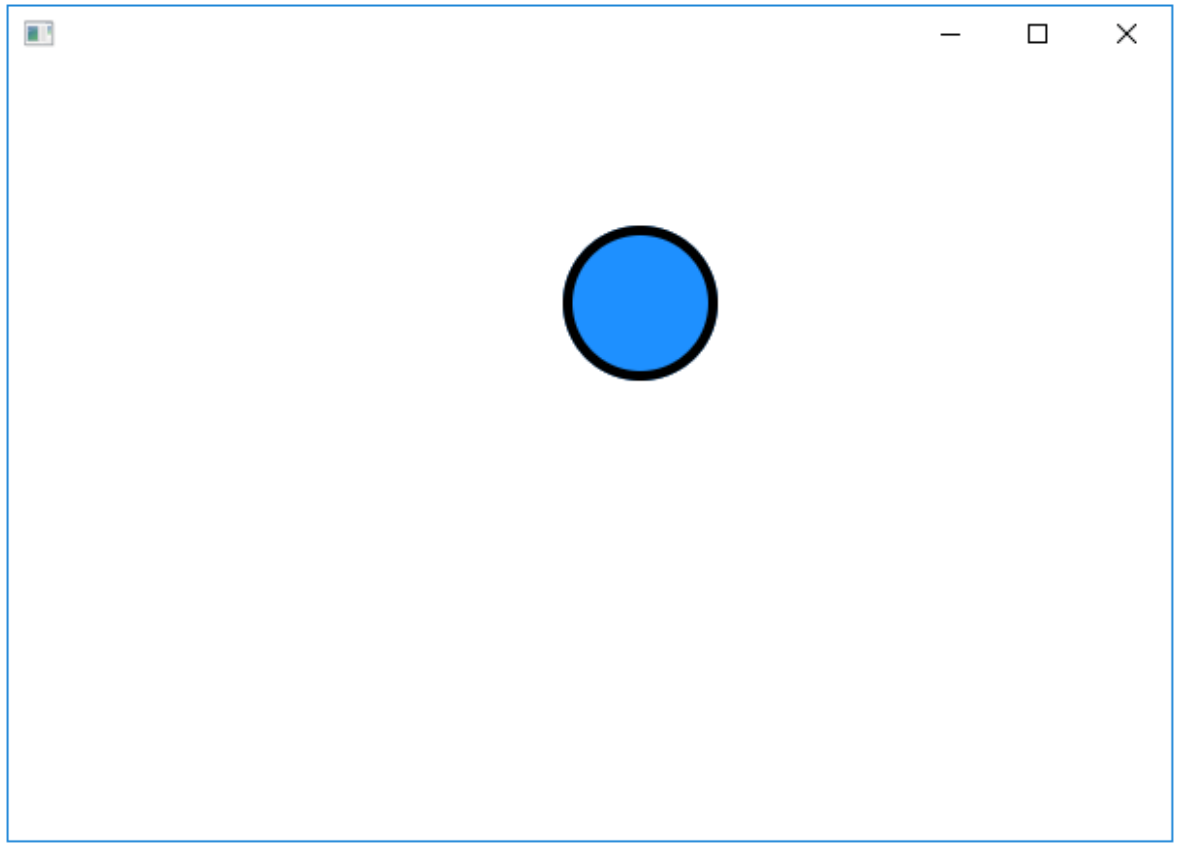


Fig. 22.13 | Bounce a circle around a window using a `Timeline` animation. (Part 5 of 6.)

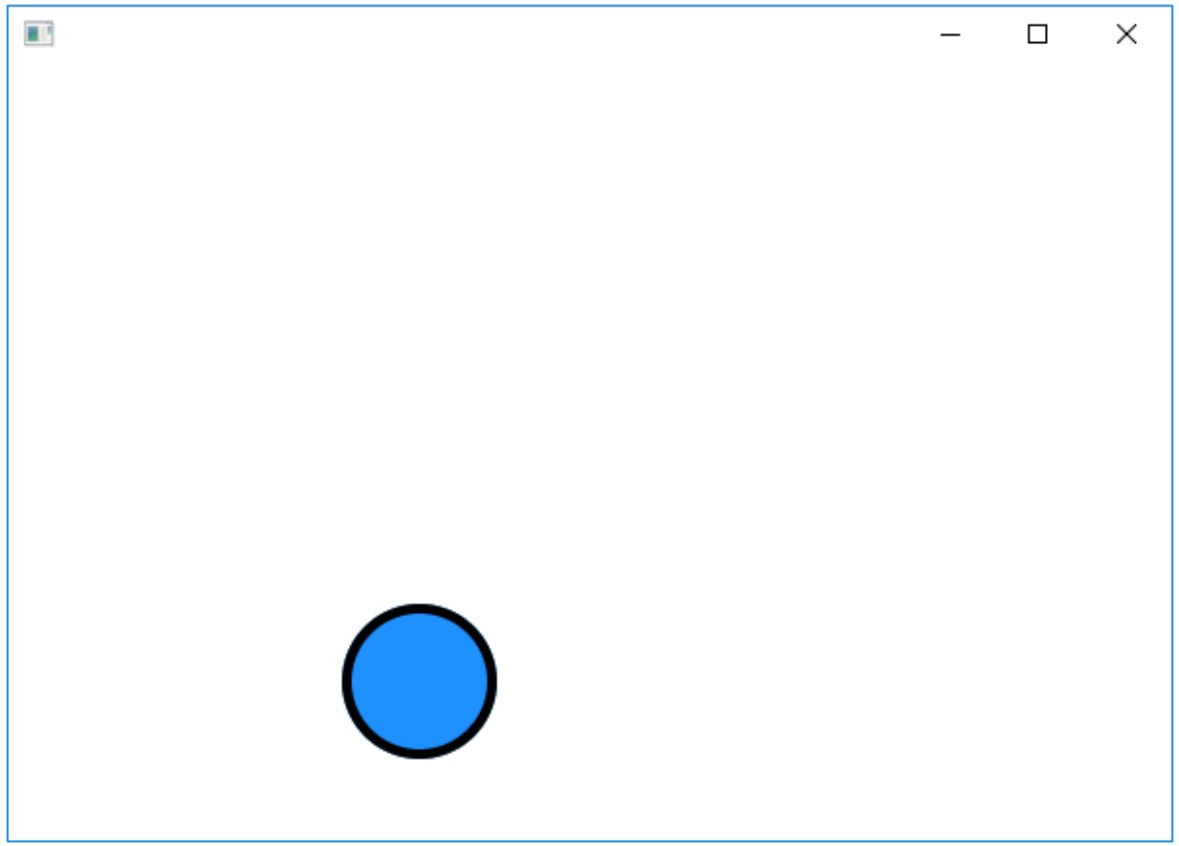


Fig. 22.13 | Bounce a circle around a window using a `Timeline` animation. (Part 6 of 6.)

```
1  // Fig. 22.14: BallAnimationTimerController.java
2  // Bounce a circle around a window using an AnimationTimer subclass.
3  import java.security.SecureRandom;
4  import javafx.animation.AnimationTimer;
5  import javafx.fxml.FXML;
6  import javafx.geometry.Bounds;
7  import javafx.scene.layout.Pane;
8  import javafx.scene.shape.Circle;
9  import javafx.util.Duration;
10
11 public class BallAnimationTimerController {
12    @FXML private Circle c;
13    @FXML private Pane pane;
14
```

Fig. 22.14 | Bounce a circle around a window using an `AnimationTimer` subclass. (Part 1 of 4.)

```
15      public void initialize() {
16         SecureRandom random = new SecureRandom();
17
18         // define a timeline animation
19         AnimationTimer timer = new AnimationTimer() {
20            int dx = 1 + random.nextInt(5);
21            int dy = 1 + random.nextInt(5);
22            int velocity = 60; // used to scale distance changes
23            long previousTime = System.nanoTime(); // time since app launch
24
25            // specify how to move Circle for current animation frame
26            @Override
27            public void handle(long now) {
28               double elapsedTime = (now - previousTime) / 1000000000.0;
29               previousTime = now;
30               double scale = elapsedTime * velocity;
```

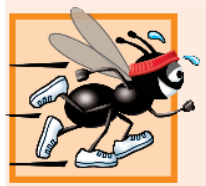
Fig. 22.14 | Bounce a circle around a window using an `AnimationTimer` subclass. (Part 2 of 4.)

```
31
32              Bounds bounds = pane.getBoundsInLocal();
33              c.setLayoutX(c.getLayoutX() + dx * scale);
34              c.setLayoutY(c.getLayoutY() + dy * scale);
35
36              if (hitRightOrLeftEdge(bounds)) {
37                 dx *= -1;
38              }
39
40              if (hitTopOrBottom(bounds)) {
41                 dy *= -1;
42              }
43           }
44        };
45
46        timer.start();
47     }
```

Fig. 22.14 | Bounce a circle around a window using an `AnimationTimer` subclass. (Part 3 of 4.)

```
48
49      // determines whether the circle hit left/right of the window
50      private boolean hitRightOrLeftEdge(Bounds bounds) {
51         return (c.getLayoutX() <= (bounds.getMinX() + c.getRadius())) ||
52            (c.getLayoutX() >= (bounds.getMaxX() - c.getRadius()));
53      }
54
55      // determines whether the circle hit top/bottom of the window
56      private boolean hitTopOrBottom(Bounds bounds) {
57         return (c.getLayoutY() <= (bounds.getMinY() + c.getRadius())) ||
58            (c.getLayoutY() >= (bounds.getMaxY() - c.getRadius()));
59      }
60   }
```

Fig. 22.14 | Bounce a circle around a window using an `AnimationTimer` subclass. (Part 4 of 4.)



Performance Tip 22.1

A `Canvas` typically is preferred for performance-oriented graphics, such as those in games with moving elements.

```
1  // Fig. 22.15: CanvasShapesController.java
2  // Drawing on a Canvas.
3  import javafx.fxml.FXML;
4  import javafx.scene.canvas.Canvas;
5  import javafx.scene.canvas.GraphicsContext;
6  import javafx.scene.image.Image;
7  import javafx.scene.paint.Color;
8  import javafx.scene.paint.CycleMethod;
9  import javafx.scene.paint.ImagePattern;
10 import javafx.scene.paint.LinearGradient;
11 import javafx.scene.paint.RadialGradient;
12 import javafx.scene.paint.Stop;
13 import javafx.scene.shape.ArcType;
14 import javafx.scene.shape.StrokeLineCap;
15
```

Fig. 22.15 | Drawing on a Canvas. (Part 1 of 5.)

```
16  public class CanvasShapesController {
17     // instance variables that refer to GUI components
18     @FXML private Canvas drawingCanvas;
19
20     // draw on the Canvas
21     public void initialize() {
22        GraphicsContext gc = drawingCanvas.getGraphicsContext2D();
23        gc.setLineWidth(10); // set all stroke widths
24
25        // draw red line
26        gc.setStroke(Color.RED);
27        gc.strokeLine(10, 10, 100, 100);
28
29        // draw green line
30        gc.setGlobalAlpha(0.5); // half transparent
31        gc.setLineCap(StrokeLineCap.ROUND);
32        gc.setStroke(Color.GREEN);
33        gc.strokeLine(100, 10, 10, 100);
34
```

Fig. 22.15 | Drawing on a Canvas. (Part 2 of 5.)

```
35          gc.setGlobalAlpha(1.0); // reset alpha transparency
36
37          // draw rounded rect with red border and yellow fill
38          gc.setStroke(Color.RED);
39          gc.setFill(Color.YELLOW);
40          gc.fillRoundRect(120, 10, 90, 90, 50, 50);
41          gc.strokeRoundRect(120, 10, 90, 90, 50, 50);
42
43          // draw circle with blue border and red/white radial gradient fill
44          gc.setStroke(Color.BLUE);
45          Stop[] stopsRadial =
46             {new Stop(0, Color.RED), new Stop(1, Color.WHITE)};
47          RadialGradient radialGradient = new RadialGradient(0, 0, 0.5, 0.5,
48             0.6, true, CycleMethod.NO_CYCLE, stopsRadial);
49          gc.setFill(radialGradient);
50          gc.fillOval(230, 10, 90, 90);
51          gc.strokeOval(230, 10, 90, 90);
52
```

Fig. 22.15 | Drawing on a Canvas. (Part 3 of 5.)

```
53          // draw ellipse with green border and image fill
54          gc.setStroke(Color.GREEN);
55          gc.setFill(new ImagePattern(new Image("yellowflowers.png")));
56          gc.fillOval(340, 10, 200, 90);
57          gc.strokeOval(340, 10, 200, 90);
58
59          // draw arc with purple border and cyan/white linear gradient fill
60          gc.setStroke(Color.PURPLE);
61          Stop[] stopsLinear =
62             {new Stop(0, Color.CYAN), new Stop(1, Color.WHITE)};
63          LinearGradient linearGradient = new LinearGradient(0, 0, 1, 0,
64             true, CycleMethod.NO_CYCLE, stopsLinear);
65          gc.setFill(linearGradient);
66          gc.fillArc(560, 10, 90, 90, 45, 270, ArcType.ROUND);
67          gc.strokeArc(560, 10, 90, 90, 45, 270, ArcType.ROUND);
68       }
69    }
```

Fig. 22.15 | Drawing on a Canvas. (Part 4 of 5.)

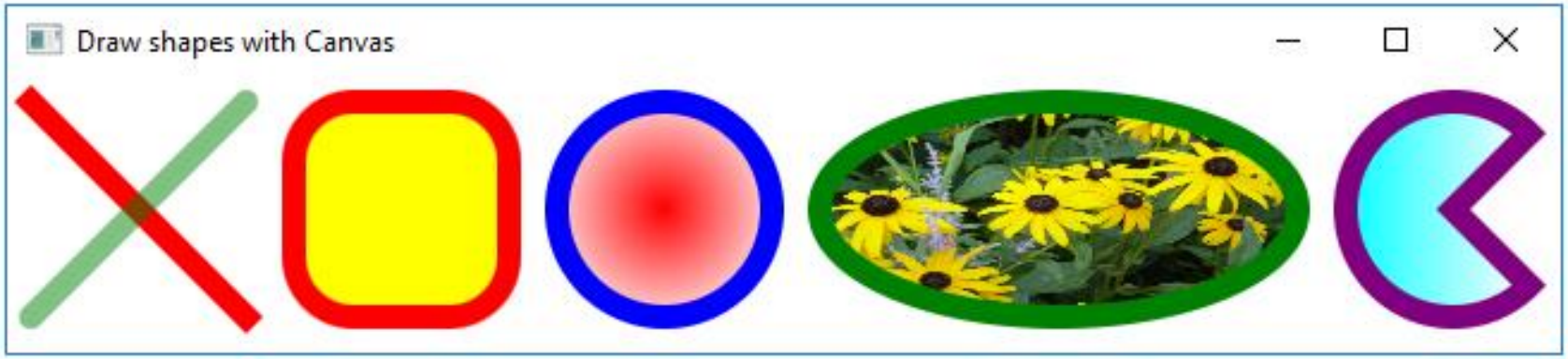


Fig. 22.15 | Drawing on a **Canvas**. (Part 5 of 5.)

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- ThreeDimensionalShapes.fxml -->
3  <!-- FXML that displays a Box, Cylinder and Sphere -->
4
5  <?import javafx.geometry.Point3D?>
6  <?import javafx.scene.layout.Pane?>
7  <?import javafx.scene.shape.Box?>
8  <?import javafx.scene.shape.Cylinder?>
9  <?import javafx.scene.shape.Sphere?>
10
11 <Pane prefHeight="200.0" prefWidth="510.0"
12    xmlns="http://javafx.com/javafx/8.0.60"
13    xmlns:fx="http://javafx.com/fxml/1"
14    fx:controller="ThreeDimensionalShapesController">
15    <children>
16       <Box fx:id="box" depth="100.0" height="100.0" layoutX="100.0"
17          layoutY="100.0" rotate="30.0" width="100.0">
18          <rotationAxis>
19             <Point3D x="1.0" y="1.0" z="1.0" />
20          </rotationAxis>
21       </Box>
```

Fig. 22.16 | FXML that displays a `Box`, `Cylinder` and `Sphere`. (Part 1 of 2.)

```
22          <Cylinder fx:id="cylinder" height="100.0" layoutX="265.0"
23             layoutY="100.0" radius="50.0" rotate="-45.0">
24             <rotationAxis>
25                <Point3D x="1.0" y="1.0" z="1.0" />
26             </rotationAxis>
27          </Cylinder>
28          <Sphere fx:id="sphere" layoutX="430.0" layoutY="100.0"
29             radius="60.0" />
30       </children>
31    </Pane>
```

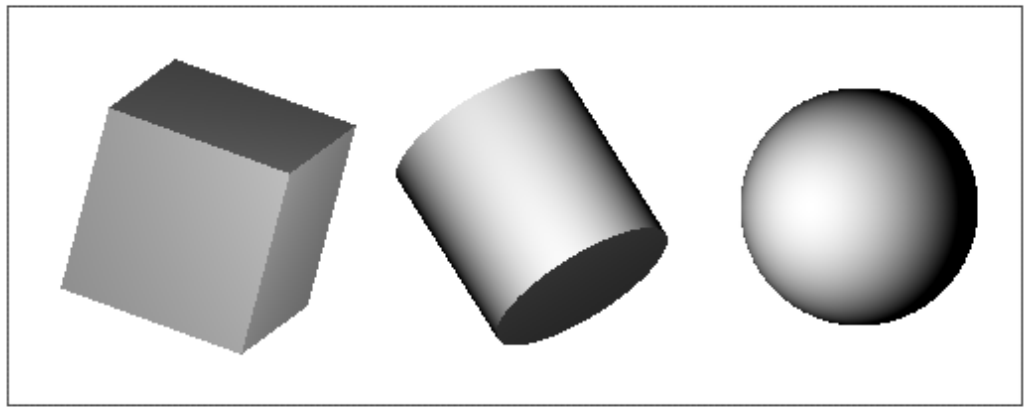


Fig. 22.16 | FXML that displays a **Box**, **Cylinder** and **Sphere**. (Part 2 of 2.)

```
1  // Fig. 22.17: ThreeDimensionalShapesController.java
2  // Setting the material displayed on 3D shapes.
3  import javafx.fxml.FXML;
4  import javafx.scene.paint.Color;
5  import javafx.scene.paint.PhongMaterial;
6  import javafx.scene.image.Image;
7  import javafx.scene.shape.Box;
8  import javafx.scene.shape.Cylinder;
9  import javafx.scene.shape.Sphere;
10
11 public class ThreeDimensionalShapesController {
12    // instance variables that refer to 3D shapes
13    @FXML private Box box;
14    @FXML private Cylinder cylinder;
15    @FXML private Sphere sphere;
16
```

Fig. 22.17 | Setting the material displayed on 3D shapes. (Part 1 of 3.)

```
17      // set the material for each 3D shape
18      public void initialize() {
19          // define material for the Box object
20          PhongMaterial boxMaterial = new PhongMaterial();
21          boxMaterial.setDiffuseColor(Color.CYAN);
22          box.setMaterial(boxMaterial);
23
24          // define material for the Cylinder object
25          PhongMaterial cylinderMaterial = new PhongMaterial();
26          cylinderMaterial.setDiffuseMap(new Image("yellowflowers.png"));
27          cylinder.setMaterial(cylinderMaterial);
28
29          // define material for the Sphere object
30          PhongMaterial sphereMaterial = new PhongMaterial();
31          sphereMaterial.setDiffuseColor(Color.RED);
32          sphereMaterial.setSpecularColor(Color.WHITE);
33          sphereMaterial.setSpecularPower(32);
34          sphere.setMaterial(sphereMaterial);
35      }
36  }
```

Fig. 22.17 | Setting the material displayed on 3D shapes. (Part 2 of 3.)

Fig. 22.17 | Setting the material displayed on 3D shapes. (Part 3 of 3.)