Chapter 22 JavaFX Graphics and Multimedia

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 Nodes while maintaining their functionality.
  - 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.
  - 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.

22.1 Introduction

- In this chapter, we continue our discussion of JavaFX from Chapters 12 and 13.

22.2 Controlling Fonts with Cascading Style Sheets (CSS)

- 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.
- Each CSS rule begins with a CSS **selector** which specifies the JavaFX objects that will be styled according to the rule.

```
.vbox {
    -fx-spacing: 10;
    -fx-padding: 10;
}

#label1 {
    -fx-font: bold 14pt Arial;
}

#label2 {
    -fx-font: 16pt "Times New Roman";
}

#label3 {
    -fx-font: bold italic 16pt "Courier New";
}

#label4 {
    -fx-font-size: 14pt;
    -fx-underline: true;
}

#label5 {
    -fx-font-size: 14pt;
}

#label5 .text {
    -fx-strikethrough: true;
}
```

a style class selector begins with a dot (.) and is followed by its **class name**

Each JavaFX CSS property name begins with -fx- followed by the name of the corresponding JavaFX object's property in all lowercase letters.

Selectors that begin with # are known as **ID selectors**—they are applied to objects with the specified ID.

The selector in this case is a combination of an ID selector and a style class selector. first locates the object with the ID label5, then within that object looks for a nested object that specifies the class text.

```
1 <?xml version="1.0" encoding="UTF-8"?>
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"
0     xmlns:fx="http://javafx.com/fxml/1">
1     <children>
2         <Label fx:id="label1" text="Arial 14pt bold" />
3         <Label fx:id="label2" text="Times New Roman 16pt plain" />
4         <Label fx:id="label3" text="Courier New 16pt bold and italic" />
5         <Label fx:id="label4" text="Default font 14pt with underline" />
6         <Label fx:id="label5" text="Default font 14pt with strikethrough"
7     </children>
8 </VBox>
```

XML declaration

comments

FXML import declarations

One root element

XML namespaces

22.2 Controlling Fonts with Cascading Style Sheets(cont.)

- Referencing the CSS File from FXML
  - 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**.

22.2 Controlling Fonts with Cascading Style Sheets(cont.)

- Programmatically Loading CSS

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

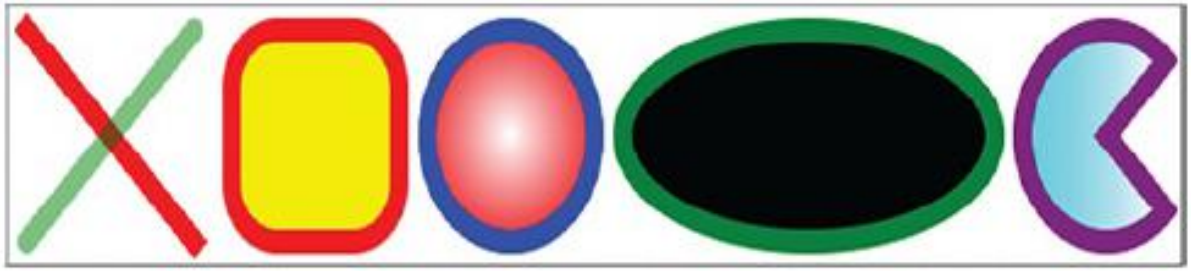
builder里面也可以绑

String java.net.URL.toExternalForm()

Constructs a string representation of this URL.

22.3 Displaying Two-Dimensional Shapes

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.

```
<Pane id="Pane" prefHeight="110.0" prefWidth="630.0"
   stylesheets="@BasicShapes.css" xmlns="http://javafx.com/javafx/8.0.60"
   xmlns:fx="http://javafx.com/fxml/1">
   <children>
      <Line fx:id="line1" endX="100.0" endY="100.0"
         startX="10.0" startY="10.0" />
      <Line fx:id="line2" endX="10.0" endY="100.0"
         startX="100.0" startY="10.0" />
      <Rectangle fx:id="rectangle" height="90.0" layoutX="120.0"
         layoutY="10.0" width="90.0" />
      <Circle fx:id="circle" centerX="270.0" centerY="55.0"
         radius="45.0" />
      <Ellipse fx:id="ellipse" centerX="430.0" centerY="55.0"
         radiusX="100.0" radiusY="45.0" />
      <Arc fx:id="arc" centerX="590.0" centerY="55.0" length="270.0"
         radiusX="45.0" radiusY="45.0" startAngle="45.0" type="ROUND" />
   </children>
</Pane>
```

```
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%, radius 60%, white, red);
27 }
28
29 Ellipse {
30     -fx-stroke: green;
31     -fx-fill: image-pattern("/ch22/BasicShapes/yellowflowers.png");
32 }
33
34 Arc {
35     -fx-stroke: purple;
36     -fx-fill: linear-gradient(to right, cyan, white);
37 }
```

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.

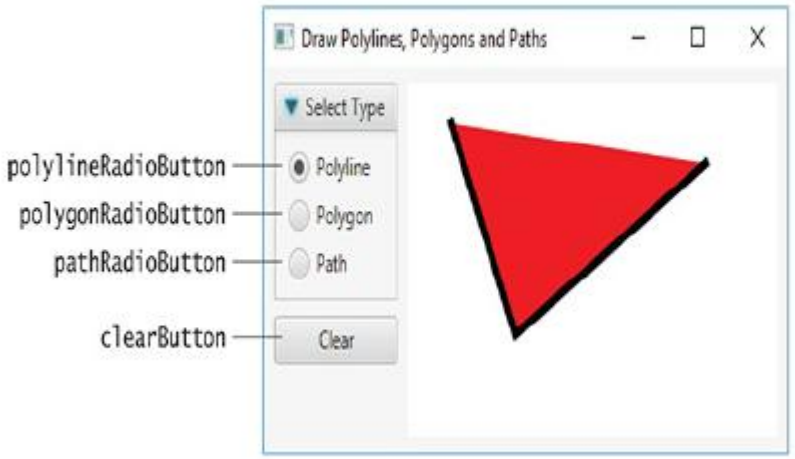
indicates that the gradient should begin from a center point at 50% 50%

This radial gradient begins with the color white in the center and ends with red at the outer edge of the Circle's fill.

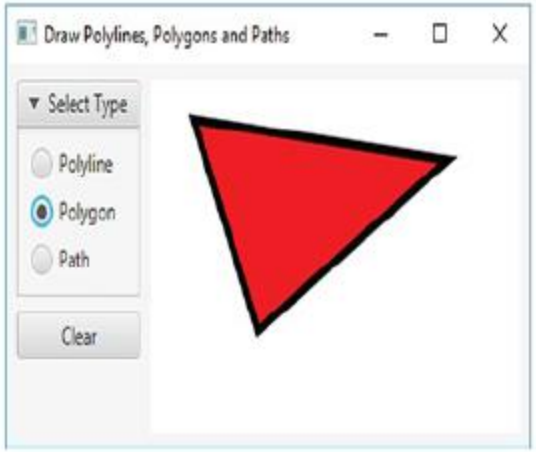
transition from one color to the next horizontally, vertically or diagonally.

```
import javafx.application.Application;

public class BasicShapes extends Application {
    @Override
    public void start(Stage stage) throws Exception {
        Parent root =
            FXMLLoader.load(getClass().getResource("BasicShapes.fxml"));

        Scene scene = new Scene(root);
        stage.setTitle("Shapes with CSS");
        stage.setScene(scene);
        stage.show();
    }

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

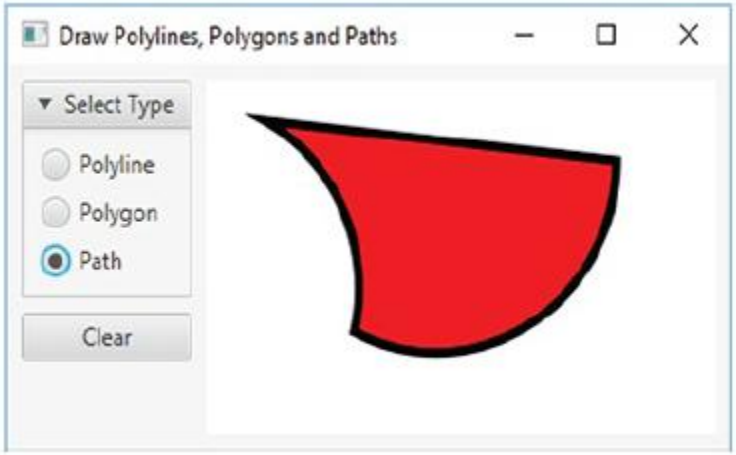
22.4 Polylines, Polygons and Paths



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



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



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

Example PolyShapes

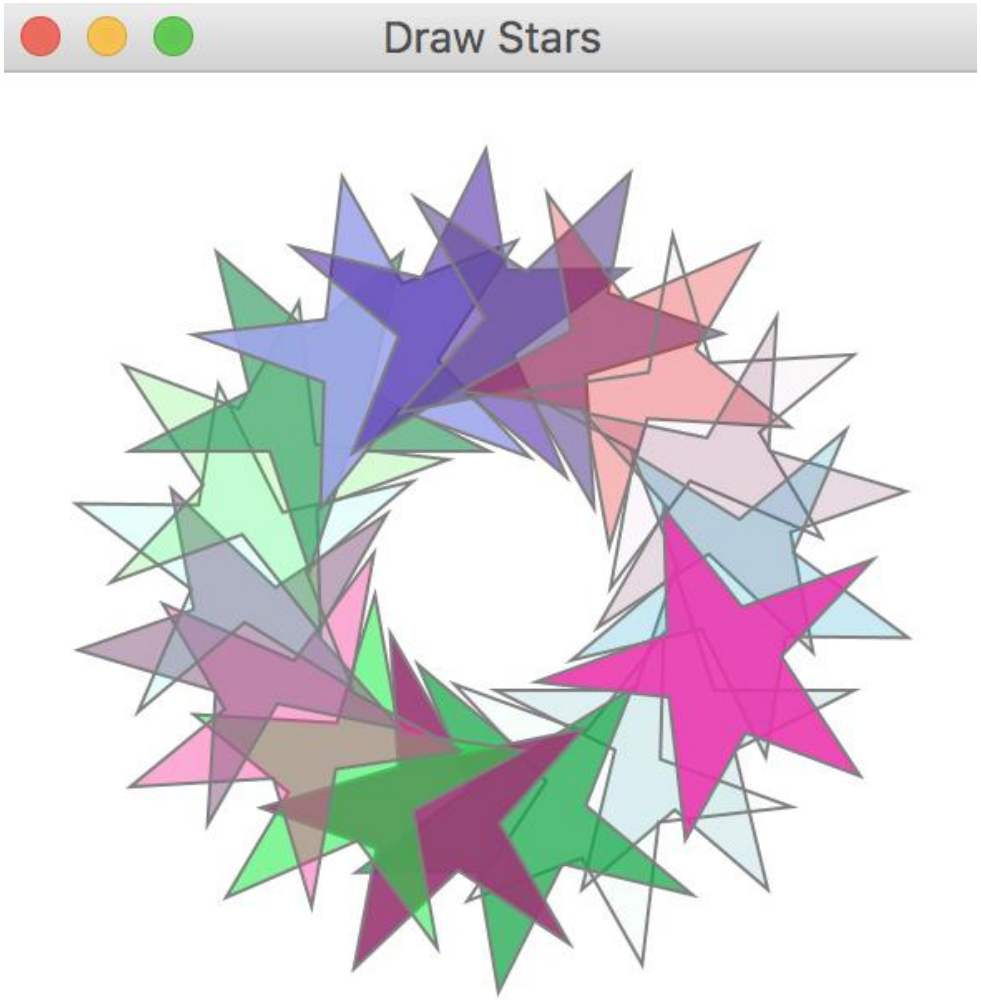
- PolyShape.css
- PolyShapes.fxml
- PolyShapes.java
- PolyShapesController.java

22.4 Polylines, Polygons and Paths(cont.)

- Method initialize
  - setUserData： associate any Object with each JavaFX control
- Method drawingAreaMouseClicked
  - MoveTo—Moves to a specific position without drawing anything.
  - ArcTo—Draws an arc from the previous PathElement's endpoint to the specified location. We'll discuss this in more detail momentarily.
  - ClosePath—Closes the path by drawing a straight line from the end point of the last PathElement to of the first PathElement

22.5 Transforms

- A transform can be applied to any UI element to reposition or reorient the element.
- The built-in JavaFX transforms are subclasses of Transform. Some of these subclasses include:
  - Translate—moves an object to a new location.
  - Rotate—rotates an object around a point and by a specified rotation angle.
  - Scale—scales an object's size by the specified amounts.


Draw Stars

```
// DrawStarsController.java
import java.security.SecureRandom;

public class DrawStarsController {
   @FXML private Pane pane;
   private static final SecureRandom random = new SecureRandom();

   public void initialize() {
      // points that define a five-pointed star shape
      Double[] points = {205.0,150.0, 217.0,186.0, 259.0,186.0,
         223.0,204.0, 233.0,246.0, 205.0,222.0, 177.0,246.0, 187.0,204.0,
         151.0,186.0, 193.0,186.0};

      // create 18 stars
      for (int count = 0; count < 18; ++count) {
         // create a new Polygon and copy existing points into it
         Polygon newStar = new Polygon();
         newStar.getPoints().addAll(points);

         // create random Color and set as newStar's fill
         newStar.setStroke(Color.GREY);
         newStar.setFill(Color.rgb(random.nextInt(255),
            random.nextInt(255), random.nextInt(255),
            random.nextDouble()));

         // apply a rotation to the shape
         newStar.getTransforms().add(
            Transform.rotate(count * 20, 150, 150));
         pane.getChildren().add(newStar);
      }
   }
```

22.6 Playing Video with Media, MediaPlayer and MediaViewer

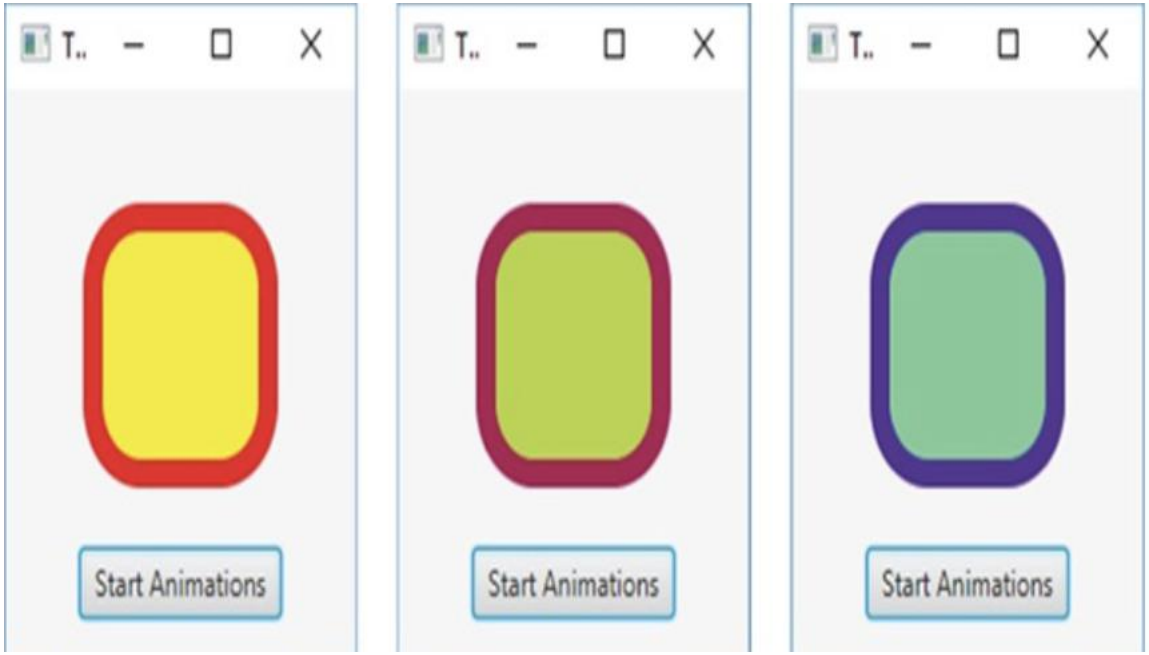
```
// VideoPlayerController.java
import java.net.URL;
import javafx.beans.binding.Bindings;
import javafx.beans.property.DoubleProperty;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;
import javafx.scene.media.MediaView;
import javafx.util.Duration;
import org.controlsfx.dialog.ExceptionDialog;

public class VideoPlayerController {
    @FXML private MediaView mediaView;
    @FXML private Button playPauseButton;
    private MediaPlayer mediaPlayer;
    private boolean playing = false;
```

```
23    public void initialize() {
24        // get URL of the video file
25        URL url = VideoPlayerController.class.getResource("sts117.mp4");
26        System.out.println(url);
27
28        // create a Media object for the specified URL
29        Media media = new Media(url.toExternalForm());
30
31        // create a MediaPlayer to control Media playback
32        mediaPlayer = new MediaPlayer(media);
33
34        // specify which MediaPlayer to display in the MediaView
35        mediaView.setMediaPlayer(mediaPlayer);
36
37        // set handler to be called when the video completes playing
38        mediaPlayer.setOnEndOfMedia(
39            new Runnable() {
40                public void run() {
41                    playing = false;
42                    playPauseButton.setText("Play");
43                    mediaPlayer.seek(Duration.ZERO);
44                    mediaPlayer.pause();
45                }
46            }
47        );
48
```

```
49          // set handler that displays an ExceptionDialog if an error occurs
50          mediaPlayer.setOnError(
51              new Runnable() {
52                  public void run() {
53                      ExceptionDialog dialog =
54                          new ExceptionDialog(mediaPlayer.getError());
55                      dialog.showAndWait();
56                  }
57              }
58          );
59
60          // set handler that resizes window to video size once ready to play
61          mediaPlayer.setOnReady(
62              new Runnable() {
63                  public void run() {
64                      DoubleProperty width = mediaView.fitWidthProperty();
65                      DoubleProperty height = mediaView.fitHeightProperty();
66                      width.bind(Bindings.selectDouble(
67                          mediaView.sceneProperty(), "width"));
68                      height.bind(Bindings.selectDouble(
69                          mediaView.sceneProperty(), "height"));
70                  }
71              }
72          );
73      }
```

```
74
75      // toggle media playback and the text on the playPauseButton
76      @FXML
77      private void playPauseButtonPressed(ActionEvent e) {
78          playing = !playing;
79
80          if (playing) {
81              playPauseButton.setText("Pause");
82              mediaPlayer.play();
83          }
84          else {
85              playPauseButton.setText("Play");
86              mediaPlayer.pause();
87          }
88      }
89  }
90
91  /*********************************************************************************************
```

22.7 Transition Animations



```
package ch22.TransitionAnimations;

// TransitionAnimationsController.java
import javafx.animation.FadeTransition;

public class TransitionAnimationsController {
    @FXML private Rectangle rectangle;

    // configure and start transition animations
    @FXML
    private void startButtonPressed(ActionEvent event) {
        // transition that changes a shape's fill
        FillTransition fillTransition =
            new FillTransition(Duration.seconds(1));
        fillTransition.setToValue(Color.CYAN);
        fillTransition.setCycleCount(2);

        // each even cycle plays transition in reverse to restore original
        fillTransition.setAutoReverse(true);
```

For this animation, during the first cycle the fill color changes from the original fill color to CYAN, and during the second cycle the animation transitions back to the original fill color.

```
// transition that changes a shape's stroke over time
StrokeTransition strokeTransition =
    new StrokeTransition(Duration.seconds(1));
strokeTransition.setToValue(Color.BLUE);
strokeTransition.setCycleCount(2);
strokeTransition.setAutoReverse(true);
```

For this animation, during the first cycle the stroke color changes from the original stroke color to BLUE, and during the second cycle the animation transitions back to the original stroke color.

```
// parallelizes multiple transitions
ParallelTransition parallelTransition =
    new ParallelTransition(fillTransition, strokeTransition);
```

In this case, the FillTransition and StrokeTransition will be performed in parallel on the app's Rectangle.

```
// transition that changes a node's opacity over time
FadeTransition fadeTransition =
    new FadeTransition(Duration.seconds(1));
fadeTransition.setFromValue(1.0); // opaque
fadeTransition.setToValue(0.0); // transparent
fadeTransition.setCycleCount(2);
fadeTransition.setAutoReverse(true);

// transition that rotates a node
```

This Transition creates a fade effect(渐变) animation that spans its duration.

```
57          // transition that rotates a node
58          RotateTransition rotateTransition =
59              new RotateTransition(Duration.seconds(1));
60          rotateTransition.setByAngle(360.0);
61          rotateTransition.setCycleCount(2);
62          rotateTransition.setInterpolator(Interpolator.EASE_BOTH);
63          rotateTransition.setAutoReverse(true);
64
65          // transition that moves a node along a Path
```

- Interpolator EASE_BOTH, which changes the rotation slowly at first (known as “easing in”), speeds up the rotation in the middle of the animation, then slows the rotation again to complete the animation

```
65        // transition that moves a node along a Path
66        Path path = new Path(new MoveTo(45, 45), new LineTo(45, 0),
67           new LineTo(90, 0), new LineTo(90, 90), new LineTo(0, 90));
68        PathTransition translateTransition =
69           new PathTransition(Duration.seconds(2), path);
70        translateTransition.setCycleCount(2);
71        translateTransition.setInterpolator(Interpolator.EASE_IN);
72        translateTransition.setAutoReverse(true);
73
74        // transition that scales a shape to make it larger or smaller
75        ScaleTransition scaleTransition =
76           new ScaleTransition(Duration.seconds(1));
77        scaleTransition.setByX(0.75);
78        scaleTransition.setByY(0.75);
79        scaleTransition.setCycleCount(2);
80        scaleTransition.setInterpolator(Interpolator.EASE_OUT);
81        scaleTransition.setAutoReverse(true);
```

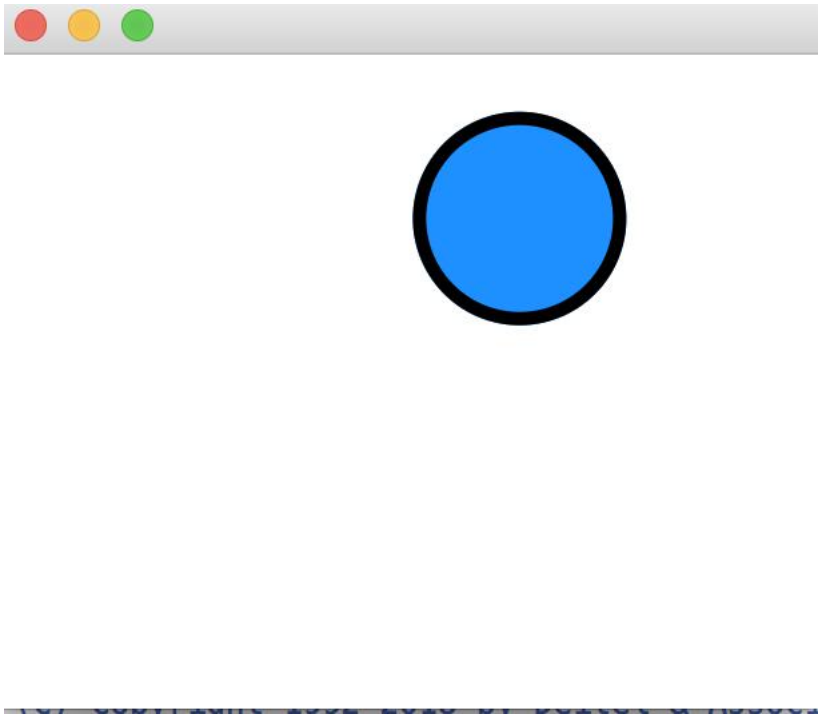
Lines 55–72 configure a two-second PathTransition that changes a shape's position by moving it along a Path.

Lines 74–81 configure a one-second ScaleTransition that changes a Node's size.

```
83          // transition that applies a sequence of transitions to a node
84          SequentialTransition sequentialTransition =
85             new SequentialTransition (rectangle, parallelTransition,
86                fadeTransition, rotateTransition, translateTransition,
87                scaleTransition);
88          sequentialTransition.play(); // play the transition
89       }
90    }
91
```

Lines 83–88 configure a `SequentialTransition` that performs a sequence of transitions—as each completes, the next one in the sequence begins executing.

22.8 Timeline Animations

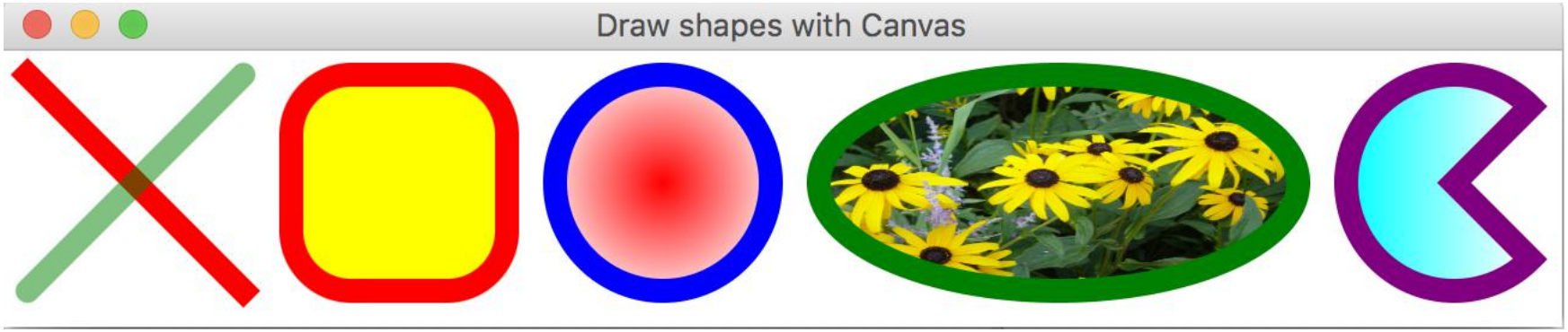


```
// define a timeline animation
Timeline timelineAnimation = new Timeline(
    new KeyFrame(Duration.millis(10),
        new EventHandler<ActionEvent>() {
            int dx = 1 + random.nextInt(5);
            int dy = 1 + random.nextInt(5);

            // move the circle by the dx and dy amounts
            @Override
            public void handle(final ActionEvent e) {
                c.setLayoutX(c.getLayoutX() + dx);
                c.setLayoutY(c.getLayoutY() + dy);
                Bounds bounds = pane.getBoundsInLocal();

                if (hitRightOrLeftEdge(bounds)) {
                    dx *= -1;
                }

                if (hitTopOrBottom(bounds)) {
                    dy *= -1;
                }
            }
        }
    )
);
```

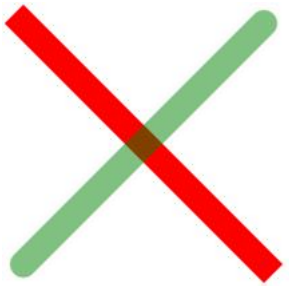
22.9 Frame-by-Frame Animation with AnimationTimer

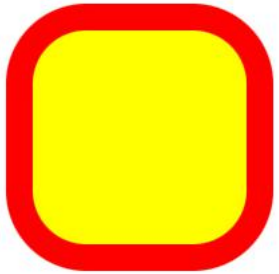
```
// define a timeline animation
AnimationTimer timer = new AnimationTimer() {
   int dx = 1 + random.nextInt(5);
   int dy = 1 + random.nextInt(5);
   int velocity = 60; // used to scale distance changes
   long previousTime = System.nanoTime(); // time since app launch

   // specify how to move Circle for current animation frame
   @Override
   public void handle(long now) {
      double elapsedTime = (now - previousTime) / 1000000000.0;
      previousTime = now;
      double scale = elapsedTime * velocity;

      Bounds bounds = pane.getBoundsInLocal();
      c.setLayoutX(c.getLayoutX() + dx * scale);
      c.setLayoutY(c.getLayoutY() + dy * scale);

      if (hitRightOrLeftEdge(bounds)) {
         dx *= -1;
      }

      if (hitTopOrBottom(bounds)) {
         dy *= -1;
      }
   }
```

22.10 Drawing on a Canvas

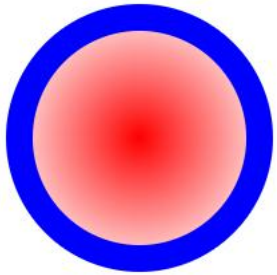


```
23 public class CanvasShapesController {
24    // instance variables that refer to GUI components
25    @FXML private Canvas drawingCanvas;
26
27    // draw on the Canvas
28    public void initialize() {
29       GraphicsContext gc = drawingCanvas.getGraphicsContext2D();
30       gc.setLineWidth(10); // set all stroke widths
31
32       // draw red line
33       gc.setStroke(Color.RED);
34       gc.strokeLine(10, 10, 100, 100);
35
36       // draw green line
37       gc.setGlobalAlpha(0.5); // half transparent
38       gc.setLineCap(StrokeLineCap.ROUND);
39       gc.setStroke(Color.GREEN);
40       gc.strokeLine(100, 10, 10, 100);
41
42       gc.setGlobalAlpha(1.0); // reset alpha transparency
43
44       // draw rounded rect with red border and yellow fill
45       gc.setStroke(Color.RED);
46       gc.setFill(Color.YELLOW);
47       gc.fillRoundRect(120, 10, 90, 90, 50, 50);
48       gc.strokeRoundRect(120, 10, 90, 90, 50, 50);
```





```
// draw circle with blue border and red/white radial gradient fill
gc.setStroke(Color.BLUE);
Stop[] stopsRadial =
   {new Stop(0, Color.RED), new Stop(1, Color.WHITE)};
RadialGradient radialGradient = new RadialGradient(0, 0, 0.5, 0.5,
   0.6, true, CycleMethod.NO_CYCLE, stopsRadial);
gc.setFill(radialGradient);
gc.fillOval(230, 10, 90, 90);
gc.strokeOval(230, 10, 90, 90);
```



```
// draw ellipse with green border and image fill
gc.setStroke(Color.GREEN);
gc.setFill(new ImagePattern(new Image(getClass().getResource("yellowflowers.png").toExternalForm())));
gc.fillOval(340, 10, 200, 90);
gc.strokeOval(340, 10, 200, 90);

// draw arc with purple border and cyan/white linear gradient fill
gc.setStroke(Color.PURPLE);
Stop[] stopsLinear =
   {new Stop(0, Color.CYAN), new Stop(1, Color.WHITE)};
LinearGradient linearGradient = new LinearGradient(0, 0, 1, 0,
   true, CycleMethod.NO_CYCLE, stopsLinear);
gc.setFill(linearGradient);
gc.fillArc(560, 10, 90, 90, 45, 270, ArcType.ROUND);
gc.strokeArc(560, 10, 90, 90, 45, 270, ArcType.ROUND);
```

22.11 Three-Dimensional Shapes

22.11 Three-Dimensional Shapes(cont.)

- Box Properties
  - Set Width, Height and Depth to 100
  - Set Layout X and Layout Y to 100 to specify the location of the cube.
  - Set Rotate to 30 to specify the rotation angle in degrees. Positive values rotate counter-clockwise.
  - For Rotation Axis, set the X, Y and Z values to 1 to indicate that the Rotate angle should be used to rotate the cube 30 degrees around each axis.

22.11 Three-Dimensional Shapes(cont.)

- Cylinder Properties
  - Set Height to 100.0 and Radius to 50.
  - Set Layout X and Layout Y to 265 and 100, respectively.
  - Set Rotate to -45 to specify the rotation angle in degrees. Negative values rotate clockwise.
  - For Rotation Axis, set the X, Y and Z values to 1 to indicate that the Rotate angle should be applied to all three axes.

22.11 Three-Dimensional Shapes(cont.)

- Sphere Properties
  - Set Radius to 60.
  - Set Layout X and Layout Y to 430 and 100, respectively.

```
public class ThreeDimensionalShapesController {
    // instance variables that refer to 3D shapes
    @FXML private Box box;
    @FXML private Cylinder cylinder;
    @FXML private Sphere sphere;
```

three-dimensional shapes

```
public void initialize() {
    // define material for the Box object
    PhongMaterial boxMaterial = new PhongMaterial();
    boxMaterial.setDiffuseColor(Color.CYAN);
    box.setMaterial(boxMaterial);

    // define material for the Cylinder object
    PhongMaterial cylinderMaterial = new PhongMaterial();
    cylinderMaterial.setDiffuseMap(new Image(
            getClass().getResource("yellowflowers.png").toExternalForm()));
    cylinder.setMaterial(cylinderMaterial);

    // define material for the Sphere object
    PhongMaterial sphereMaterial = new PhongMaterial();
    sphereMaterial.setDiffuseColor(Color.RED);
    sphereMaterial.setSpecularColor(Color.WHITE);
    sphereMaterial.setSpecularPower(32);
    sphere.setMaterial(sphereMaterial);
```