

22.4 Polylines, Polygons and Paths

There are several kinds of JavaFX shapes that enable you to create custom shapes:

- **Polyline**—draws a series of connected lines defined by a set of points.
- **Polygon**—draws a series of connected lines defined by a set of points and connects the last point to the first point.
- **Path**—draws a series of connected **PathElements** by moving to a given point, then drawing lines, arcs and curves.

In the **PolyShapes** app, you select which shape you want to display by selecting one of the **RadioButtons** in the left column. You specify a shape's points by clicking throughout the **AnchoredPane** in which the shapes are displayed.

For this example, we do not show the **PolyShapes** subclass of **Application** (located in the example's **PolyShapes.java** file), because it loads the FXML and displays the GUI, as demonstrated in [Chapters 12](#) and [13](#).

22.4.1 GUI and CSS

This app's GUI ([Fig. 22.5](#)) is similar to that of the **Painter** app in [Section 13.3](#). For that reason, we show only the key

GUI elements' `fx:id` property values, rather than the complete FXML—each `fx:id` property value ends with the GUI element's type. In this GUI:

- The three `RadioButton`s are part of a `ToggleGroup` with the `fx:id` "toggleGroup". The **Polyline** `RadioButton` should be **Selected** by default. We also set each `RadioButton`'s **On Action** event handler to `shapeRadioButtonSelected`.
- We dragged a `Polyline`, a `Polygon` and a `Path` from the Scene Builder **Library's Shapes** section onto the `Pane` that displays the shapes, and we set their `fx:ids` to `polyline`, `polygon` and `path`, respectively. We set each shape's `visible` property to `false` by selecting the shape in Scene Builder, then unchecking the **Visible** checkbox in the **Properties** inspector. We display only the shape with the selected `RadioButton` at runtime.
- We set the `Pane`'s **On Mouse Clicked** event handler to `drawingAreaMouseClicked`.
- We set the **Clear** Button's **On Action** event handler to `clearButtonPressed`.
- We set the controller class to `PolyShapesController`.
- Finally, we edited the FXML to remove the `Path` object's `<elements>` and the `Polyline` and `Polygon` objects' `<points>`, as we'll set these programmatically in response to the user's mouse-click events.

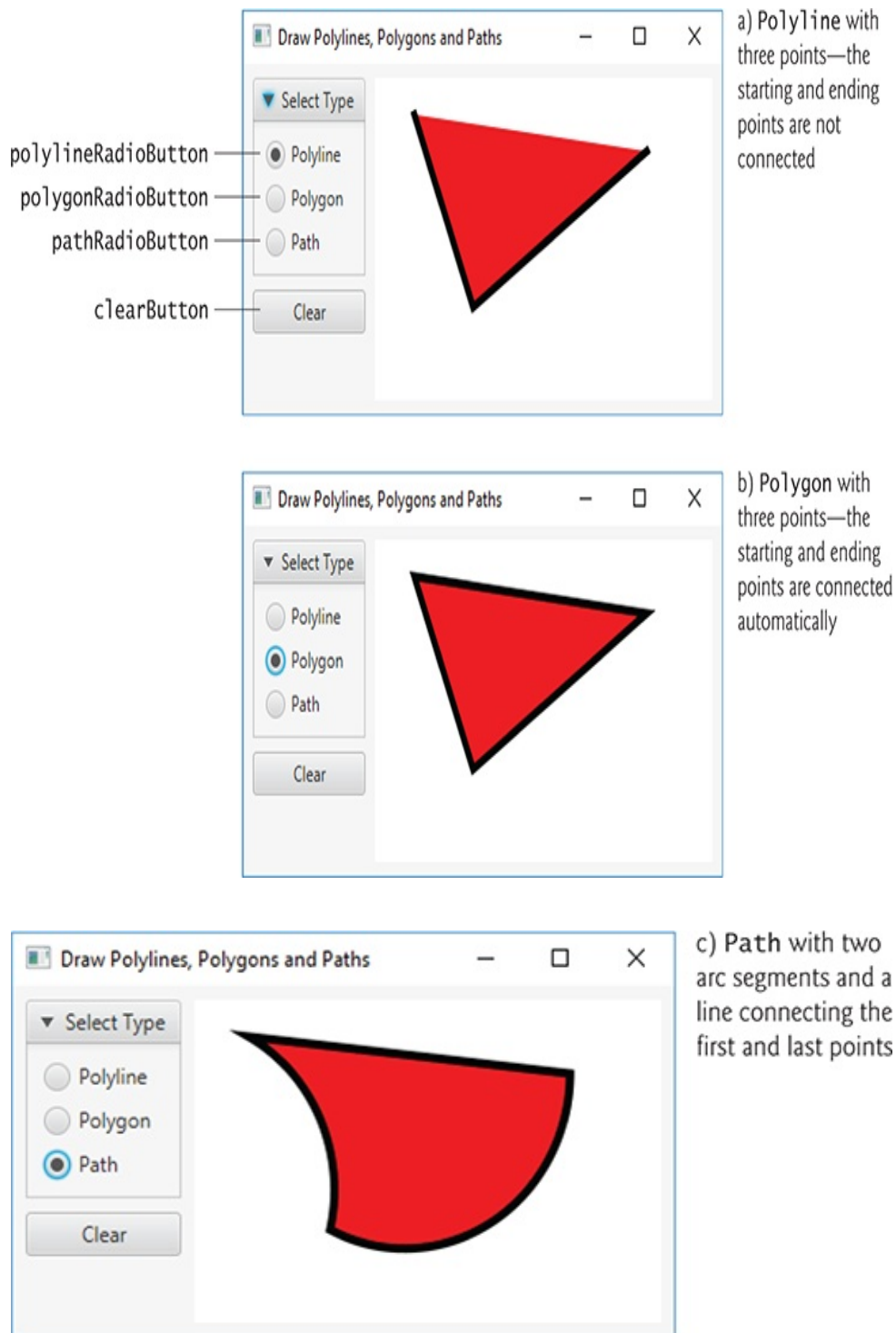


Fig. 22.5

Polylines, Polygons and Paths.

Description

The `PolyShapes.css` file defines the properties `-fx-stroke`, `-fx-stroke-width` and `-fx-fill` that are applied to all three shapes in this example. As you can see in [Fig. 22.5](#), the stroke is a thick black line (5 pixels wide) and the fill is red.

```
Polyline, Polygon, Path {  
    -fx-stroke: black;  
    -fx-stroke-width: 5;  
    -fx-fill: red;  
}
```

22.4.2 PolyShapesController Class

[Figure 22.6](#) shows this app's `PolyShapesController` class, which responds to the user's interactions. The `enum ShapeType` (line 17) defines three constants that we use to determine which shape to display. Lines 20–26 declare the variables that correspond to the GUI components and shapes with `fx:ids` in the FXML. The `shapeType` variable (line 29) stores whichever shape type is currently selected in the GUI's `RadioButtons`—by default, the `Polyline` will be

displayed. As you'll soon see, the `sweepFlag` variable is used to determine whether an arc in a `Path` is drawn with a negative or positive sweep angle.

```
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
15 public class PolyShapesController {
16     // enum representing shape types
17     private enum ShapeType {POLYLINE, POLYGON, PA
18
19     // instance variables that refer to GUI compo
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.POLYL
30     private boolean sweepFlag = true; // used wit
31
32     // set user data for the RadioButtons and dis
33     public void initialize() {
34         // user data on a control can be any Objec
```

```

35     polylineRadioButton.setUserData(ShapeType.
36     polygonRadioButton.setUserData(ShapeType.P
37     pathRadioButton.setUserData(ShapeType.PATH
38
39     displayShape(); // sets polyline's visibil
40     }
41
42     // handles drawingArea's onMouseClicked event
43     @FXML
44     private void drawingAreaMouseClicked(MouseEve
45     polyline.getPoints().addAll(e.getX(), e.ge
46     polygon.getPoints().addAll(e.getX(), e.get
47
48     // if path is empty, move to first click p
49     if (path.getElements().isEmpty()) {
50     path.getElements().add(new MoveTo(e.get
51     path.getElements().add(new ClosePath())
52     }
53     else { // insert a new path segment before
54     // create an arc segment and insert it
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
63     }
64     }
65
66     // handles color RadioButton's ActionEvents
67     @FXML
68     private void shapeRadioButtonSelected(ActionE
69     // user data for each color RadioButton is
70     shapeType =
71     (ShapeType) toggleGroup.getSelectedTogg
72     displayShape(); // display the currently s
73     }
74

```

```

75      // displays currently selected shape
76      private void displayShape() {
77          polyline.setVisible(shapeType == ShapeType
78          polygon.setVisible(shapeType == ShapeType.
79          path.setVisible(shapeType == ShapeType.PAT
80      }
81
82      // resets each shape
83      @FXML
84      private void clearButtonPressed(ActionEvent e
85          polyline.getPoints().clear();
86          polygon.getPoints().clear();
87          path.getElements().clear();
88      }
89  }

```

Fig. 22.6

Drawing Polylines, Polygons and Paths.

Method initialize

Recall from [Section 13.3.1](#) that you can associate any `Object` with each JavaFX control via its `setUserData` method. For the shape `RadioButtons` in this app, we store the specific `ShapeType` that the `RadioButton` represents (lines 35–37). We use these values when handling the `RadioButton` events to set the `shapeType` instance variable. Line 39 then calls method `displayShape` to display the currently selected shape (the `Polyline` by

default). Initially, the shape is not visible because it does not yet have any points.

Method `drawingAreaMouseClicked`

When the user clicks the app's `Pane`, method `drawingAreaMouseClicked` (lines 43–64) modifies all three shapes to incorporate the new point at which the user clicked. `Polyline`s and `Polygon`s store their points as a collection of `Double` values in which the first two values represent the first point's location, the next two values represent the second point's location, etc. Line 45 gets the `polyline` object's collection of points, then adds the new click point to the collection by calling its `addAll` method and passing the `MouseEvent`'s *x*- and *y*-coordinate values. This adds the new point's information to the end of the collection. Line 46 performs the same task for the `polygon` object.

Lines 49–63 manipulate the `path` object. A `Path` is represented by a collection of `PathElements`. The subclasses of `PathElement` used in this example are:

- `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 the start point of the first `PathElement`.

Other `PathElements` not covered here include `LineTo`, `HLineTo`, `VLineTo`, `CubicCurveTo` and `QuadCurveTo`.

When the user clicks the `Pane`, line 49 checks whether the `Path` contains elements. If not, line 50 moves the starting point of the path to the mouse-click location by adding a `MoveTo` element to the path's `PathElements` collection. Then line 51 adds a new `ClosePath` element to complete the path. For each subsequent mouse-click event, lines 55–60 create an `ArcTo` element and line 62 inserts it before the `ClosePath` element by calling the `PathElements` collection's `add` method that receives an index as its first argument.

Lines 56–57 set the `ArcTo` element's end point to the `MouseEvent`'s coordinates. The arc is drawn as a piece of an ellipse for which you specify the horizontal radius and vertical radius (lines 58–59). Line 60 sets the `ArcTo`'s `sweepFlag`, which determines whether the arc sweeps in the positive angle direction (`true`; counter clockwise) or the negative angle direction (`false`; clockwise). By default an `ArcTo` element is drawn as the shortest arc between the last `PathElement`'s end point and the point specified by the `ArcTo` element. To sweep the arc the long way around the ellipse, set the `ArcTo`'s `largeArcFlag` to `true`. For each mouse click, line 61 reverses the value of our controller class's `sweepFlag`

instance variable so that the `ArcTo` elements toggle between positive and negative angles for variety.

Method `shapeRadioButtonSelected`

When the user clicks a shape `RadioButton`, lines 70–71 set the controller’s `shapeType` instance variable, then line 72 calls method `displayShape` to display the selected shape. Try creating a `Polyline` of several points, then changing to the `Polygon` and `Path` to see how the points are used in each shape.

Method `displayShape`

Lines 77–79 simply set the visibility of the three shapes, based on the current `shapeType`. The currently selected shape’s visibility is set to `true` to display the shape, and the other shapes’ visibility is set to `false` to hide those shapes.

Method `clearButtonPressed`

When the user clicks the **Clear Button**, lines 85–86 clear the `polyline`'s and `polygon`'s collections of points, and line 87 clears the `path`'s collection of `PathElements`. The user can then begin drawing a new shape by clicking the **Pane**.