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 `RadioButton`s in the left column. You specify a shape's points by clicking throughout the `AnchoredPane` in which the shapes are displayed.

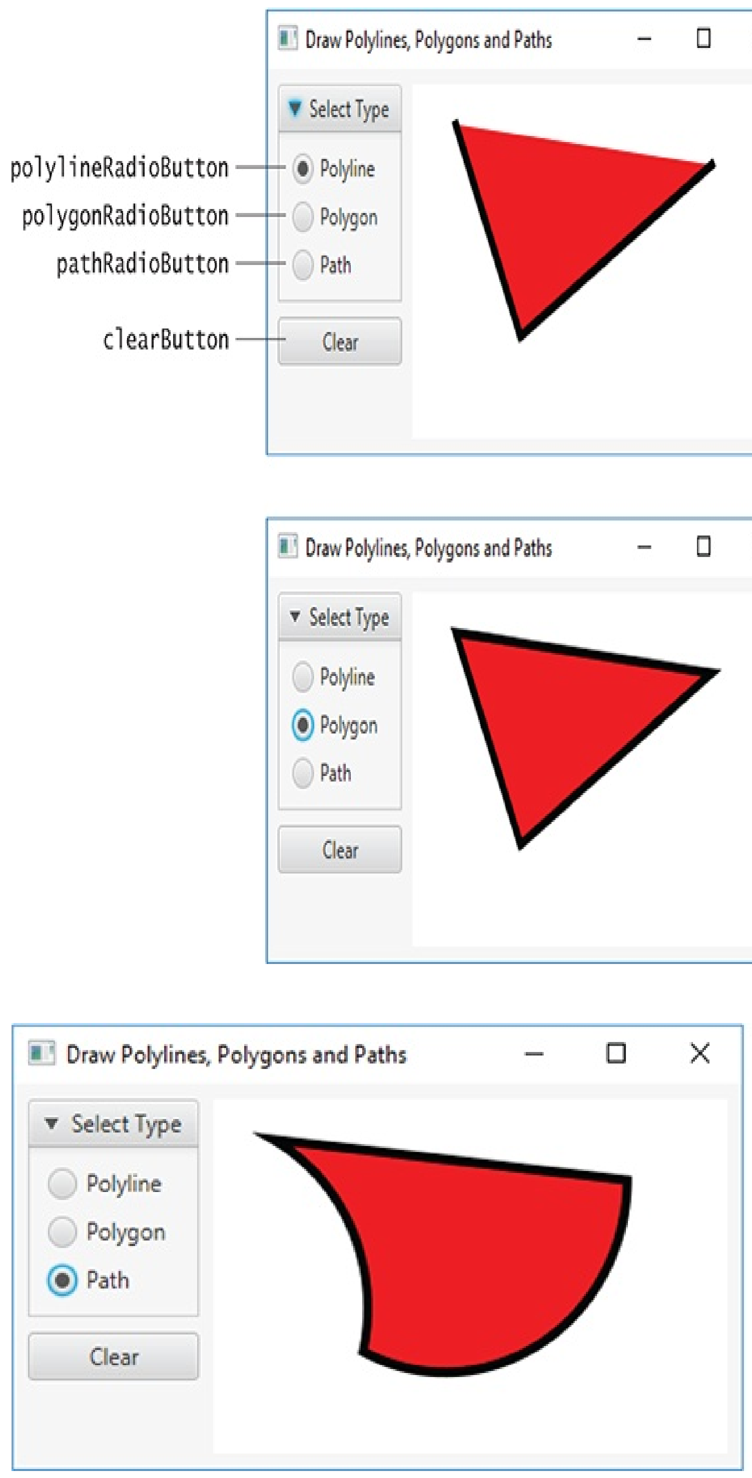
For this example, we do not show the `PolyShapes` subclass of `Application` (located in the example's `PolyShapes.java` file), because it loads the FXML and displays the GUI, as demonstrated in Chapters 12 and 13.

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:id`s 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.



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

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.

```
// Fig. 22.6: PolyShapesController.java
// Drawing Polylines, Polygons and Paths.
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.RadioButton;
import javafx.scene.control.ToggleGroup;
import javafx.scene.input.MouseEvent;
import javafx.scene.shape.ArcTo;
import javafx.scene.shape.ClosePath;
import javafx.scene.shape.MoveTo;
import javafx.scene.shape.Path;
import javafx.scene.shape.Polygon;
import javafx.scene.shape.Polyline;

public class PolyShapesController {
   // enum representing shape types
   private enum ShapeType {POLYLINE, POLYGON, PA

   // instance variables that refer to GUI compo
   @FXML private RadioButton polylineRadioButton
   @FXML private RadioButton polygonRadioButton;
   @FXML private RadioButton pathRadioButton;
   @FXML private ToggleGroup toggleGroup;
   @FXML private Polyline polyline;
   @FXML private Polygon polygon;
   @FXML private Path path;

   // instance variables for managing state
   private ShapeType shapeType = ShapeType.POLYL
   private boolean sweepFlag = true; // used wit

   // set user data for the RadioButtons and dis
   public void initialize() {
      // user data on a control can be any Objec
```

```java
      polylineRadioButton.setUserData(ShapeType.
      polygonRadioButton.setUserData(ShapeType.P
      pathRadioButton.setUserData(ShapeType.PATH

      displayShape(); // sets polyline's visibil
   }

   // handles drawingArea's onMouseClicked event
   @FXML
   private void drawingAreaMouseClicked(MouseEve
      polyline.getPoints().addAll(e.getX(), e.ge
      polygon.getPoints().addAll(e.getX(), e.get

      // if path is empty, move to first click p
      if (path.getElements().isEmpty()) {
         path.getElements().add(new MoveTo(e.get
         path.getElements().add(new ClosePath())
      }
      else { // insert a new path segment before
         // create an arc segment and insert it
         ArcTo arcTo = new ArcTo();
         arcTo.setX(e.getX());
         arcTo.setY(e.getY());
         arcTo.setRadiusX(100.0);
         arcTo.setRadiusY(100.0);
         arcTo.setSweepFlag(sweepFlag);
         sweepFlag = !sweepFlag;
         path.getElements().add(path.getElements
      }
   }

   // handles color RadioButton's ActionEvents
   @FXML
   private void shapeRadioButtonSelected(ActionE
      // user data for each color RadioButton is
      shapeType =
         (ShapeType) toggleGroup.getSelectedTogg
      displayShape(); // display the currently s
   }

```

```
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 `Polyline`s, `Polygon`s and `Path`s.

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 `RadioButton`s 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 `PathElement`s. 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 `PathElement`s. The user can then begin drawing a new shape by clicking the `Pane`.