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.

The next example draws stars using the `Polygon` control and uses `Rotate` transforms to create a circle of randomly colored stars. The FXML for this app consists of an empty 300-by-300 `Pane` layout with the `fx:id "pane"`. We also set the controller class to `DrawStarsController`. Figure 22.7 shows the app's controller and a sample output.

Method initialize (lines 14–37) defines the stars, applies the transforms and attaches the stars to the app's `pane`. Lines 16–18 define the points of a star as an array of type `Double`—the collection of points stored in a `Polygon` is implemented with a generic collection, so you must use type `Double` rather than `double` (recall that primitive types cannot be used in Java generics). Each pair of values in the array represents the *x*- and *y*-coordinates of one point in the `Polygon`. We defined ten points in the array.

```
1   // Fig. 22.7: DrawStarsController.java
2   // Create a circle of stars using Polygons and R
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 = ne
13
14     public void initialize() {
15        // points that define a five-pointed star
16        Double[] points = {205.0,150.0, 217.0,186.
17           223.0,204.0, 233.0,246.0, 205.0,222.0,
18           223.0,204.0, 233.0,246.0, 205.0,222.0,
19
20        // create 18 stars
21        for (int count = 0; count < 18; ++count) {
22           // create a new Polygon and copy existi
23           Polygon newStar = new Polygon();
24           newStar.getPoints().addAll(points);
25
26           // create random Color and set as newSt
27           newStar.setStroke(Color.GREY);
28           newStar.setFill(Color.rgb(random.nextIn
29              random.nextInt(255), random.nextInt(
30                 random.nextDouble()));
31
32           // apply a rotation to the shape
33           newStar.getTransforms().add(
34              Transform.rotate(count * 20, 150, 15
35           pane.getChildren().add(newStar);
36        }
37     }
38  }
```

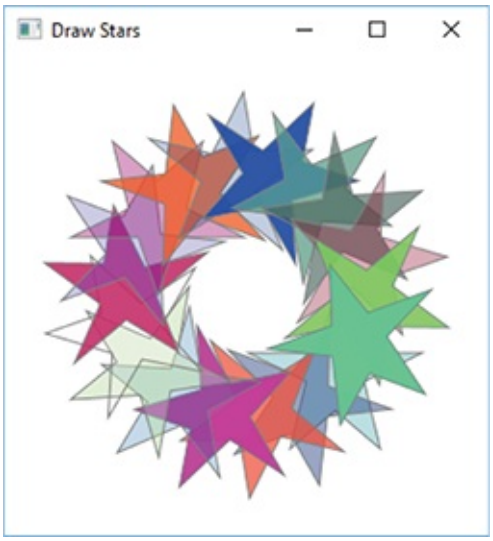

Fig. 22.7

Create a circle of stars using `Polygon`s and `Rotate` transforms.

Description

During each iteration of the loop, lines 23–34 create a `Polygon` using the points in the `points` array and apply a different `Rotate` transform. This results in the circle of `Polygon`s in the screen capture. To generate the random colors for each star, we use a `Secure-Random` object to create three random values from 0–255 for the red, green and blue components of the color, and one random value from 0.0–1.0 for the color's alpha transparency value. We pass those values to class `Color`'s `static rgb` method to create a `Color`.

To apply a rotation to the new `Polygon`, we add a `Rotate`

transform to the `Polygon`'s collection of `Transform`s (lines 33–34). To create the `Rotate` transform object, we invoke class `Transform`'s `static` method `rotate` (line 34), which returns a `Rotate` object. The method's first argument is the rotation angle. Each iteration of the loop assigns a new rotation-angle value by using the control variable multiplied by 20 as the `rotate` method's first argument. The method's next two arguments are the *x*- and *y*-coordinates of the point of rotation around which the `Polygon` rotates. The center of the circle of stars is the point *(150, 150)*, because we rotated all 18 stars around that point. Adding each `Polygon` as a new child element of the `pane` object allows the `Polygon` to be rendered on screen.