22.8 Timeline Animations

In this section, we continue our animation discussion with a `Timeline` animation that bounces a `Circle` object around the app's `Pane` over time. A `Timeline` animation can change any `Node` property that's modifiable. You specify how to change property values with one or more `KeyFrame` objects that the `Timeline` animation performs in sequence. For this app, we'll specify a single `KeyFrame` that modifies a `Circle`'s location, then we'll play that `KeyFrame` indefinitely. Figure 22.12 shows the app's FXML, which defines a `Circle` object with a five-pixel black border and the fill color `DODGERBLUE`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Fig. 22.12: TimelineAnimation.fxml -->
<!-- FXML for a Circle that will be animated by

<?import javafx.scene.layout.Pane?>
<?import javafx.scene.shape.Circle?>

<Pane id="Pane" fx:id="pane" prefHeight="400.0"
   prefWidth="600.0" xmlns:fx="http://javafx.com
   xmlns="http://javafx.com/javafx/8.0.60"
   fx:controller="TimelineAnimationController">
   <children>
      <Circle fx:id="c" fill="DODGERBLUE" layout
         radius="40.0" stroke="BLACK" strokeType
         strokeWidth="5.0" />
   </children>
```

```
</Pane>
```

Fig. 22.12

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

The application’s controller (Fig. 22.13) configures then plays the `Timeline` animation in the `initialize` method. Lines 22–45 define the animation, line 48 specifies that the animation should cycle indefinitely (until the program terminates or the animation’s `stop` method is called) and line 49 plays the animation.

```
1   // Fig. 22.13: TimelineAnimationController.java
2   // Bounce a circle around a window using a Timel
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
14  public class TimelineAnimationController {
15     @FXML Circle c;
16     @FXML Pane pane;
17
18     public void initialize() {
19        SecureRandom random = new SecureRandom();
```

```

      // 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
               @Override
               public void handle(final ActionEv
                  c.setLayoutX(c.getLayoutX() +
                  c.setLayoutY(c.getLayoutY() +
                  Bounds bounds = pane.getBounds

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

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

      // indicate that the timeline animation sh
      timelineAnimation.setCycleCount(Timeline.I
      timelineAnimation.play();
   }

   // determines whether the circle hit the left
   private boolean hitRightOrLeftEdge(Bounds bou
      return (c.getLayoutX() <= (bounds.getMinX(
         (c.getLayoutX() >= (bounds.getMaxX() -
   }

   // determines whether the circle hit the top
   private boolean hitTopOrBottom(Bounds bounds)
```

```
         return (c.getLayoutY() <= (bounds.getMinY(
            (c.getLayoutY() >= (bounds.getMaxY() -
      }
   }
```

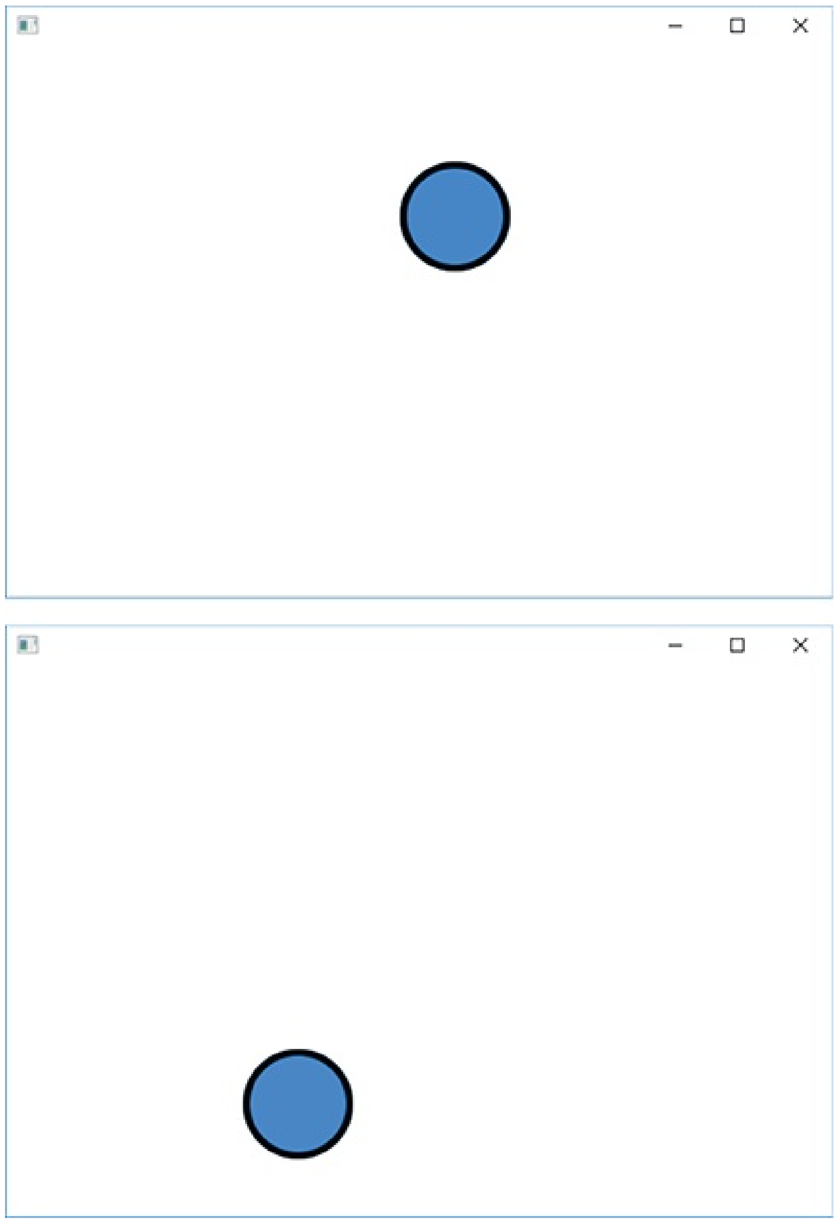

Fig. 22.13

Bounce a circle around a window using a `Timeline` animation.

Description

Creating the `Timeline`

The `Timeline` constructor used in lines 22–45 can receive a comma-separated list of `KeyFrame`s as arguments—in this case, we pass a single `KeyFrame`. Each `KeyFrame` issues an `ActionEvent` at a particular time in the animation. The app can respond to the event by changing a `Node`'s property values. The `KeyFrame` constructor used here specifies that, after 10 milliseconds, the `ActionEvent` will occur. Because we set the `Timeline`'s cycle count to `Timeline.INDEFINITE`, the `Timeline` will perform this `KeyFrame` every 10 milliseconds. Lines 24–43 define the `EventHandler` for the `KeyFrame`'s `ActionEvent`.

`KeyFrame`'s `EventHandler`

In the `KeyFrame`'s `EventHandler` we define instance variables `dx` and `dy` (lines 25–26) and initialize them with randomly chosen values that will be used to change the `Circle`'s *x*- and *y*-coordinates each time the `KeyFrame` plays. The `EventHandler`'s `handle` method (lines 29–42) adds these values to the `Circle`’s *x*- and *y*-coordinates (lines 31–32). Next, lines 35–41 perform bounds checking to determine whether the `Circle` has collided with any of the `Pane`’s edges. If the `Circle` hits the left or right edge, line 36 multiplies the value of `dx` by `-1` to reverse the `Circle`’s horizontal direction. If the `Circle` hits the top or bottom edge, line 40 multiplies the value of `dx` by `-1` to reverse the `Circle`’s horizontal direction.