The JavaFX Canvas – Part I: Drawing

This tutorial will focus on using the canvas for drawing on screen. To complete it make sure that you know how to create a basic JavaFX window as detailed in the first JavaFX tutorial on yatishparmar.com: Creating a JavaFX window.

The canvas provides a blank surface that you can customise. Because it is a subclass of node you can use it in the JavaFX graph. This means that you can add it to Group or any other layout. To use a Canvas you must first create a Canvas object, get its GraphicsContext and then calling various different draw operations.

Creating A Basic Drawing

Create a new Java project, add the necessary imports and create a basic JavaFX window. Use a Group as the root. Create a canvas and add this to your root Group. Here is the basic code for my Canvas tutorial.

```
1   package canvastutorial;
2
3   import javafx.application.Application;
4   import javafx.scene.Group;
5   import javafx.scene.Scene;
6   import javafx.scene.canvas.Canvas;
7   import javafx.stage.Stage;
8
9   public class CanvasTutorial extends Application {
10
11
12      public static void main(String[] args) {
13          launch(args);
14      }
15
16      @Override
17      public void start(Stage primaryStage) throws Exception {
18          Group g = new Group();
19          Scene scene = new Scene(g);
20          Canvas canvas = new Canvas(300,300);
21          g.getChildren().add(canvas);
22
23          primaryStage.setTitle("Canvas test");
24          primaryStage.setScene(scene);
25          primaryStage.show();
26      }
27
28  }
```

To draw on the Canvas I need to retrieve its GraphicsContext. I can then use this to draw shapes and write text, add effects to these, or paint images. I do that by adding

```
GraphicsContext gc = canvas.getGraphicsContext2D();
```

You will have to import `javafx.scene.canvas.GraphicsContext` to do this.

Drawing On The Canvas

In order to draw on the canvas you will have to:

1. Set a fill colour (if you want a solid shape
2. Set a line colour (if an outline is needed)
3. Draw your shapes

Here is an example of drawing a red rectangle that covers the whole window specified above:

```
gc.setFill(Color.RED);
gc.fillRect(0, 0, 300, 300);
```

If I wanted to clear the whole window I would need to:

```
gc.clearRect(0, 0, 300, 300);
```

In comparison this would clear a rectangle 100 pixels by 50 pixels in the middle of the screen:

```
gc.clearRect(0, 0, 300, 300);
```

You can also try the following code to see what happens:

```
gc.setFill(Color.CRIMSON);
gc.setStroke(Color.BLACK);
gc.setLineWidth(5);
gc.strokeLine(50, 20, 20, 50);
gc.strokeArc(60, 160, 30, 30, 45, 240, ArcType.CHORD);
gc.strokeArc(110, 160, 30, 30, 45, 240, ArcType.OPEN);
```

It is worth experimenting with some of the different drawing properties such as `fillText` and `strokeText` to see what happens.

This example will draw a rectangle that is filled with a gradient:

```
gc.setFill(new LinearGradient(1, 1, 0, 0, true,
    CycleMethod.REFLECT,
    new Stop(0, Color.CORNFLOWERBLUE),
    new Stop(1, Color.LIGHTBLUE)));

gc.fillRect(0, 0, 300, 300);
```

JavaFX Animations

JavaFX makes it easy to animate objects in a root layout. The example below:

1. Draws an orange square.
2. Instantiates an object called translate from the TranslateTransition class. This moves the square across the layout within a specified time.
3. Sets the x and y co-ordinates that the square will move to using the translate object.
4. Creates a transition object which will apply the specified transition to the specified objects.
5. The transition is set to run indefinitely, returning to the start position.
6. Finally the transition is played.

```
Rectangle r = new Rectangle(0,0,30, 30);
r.setFill(Color.ORANGE);
g.getChildren().add(r);

TranslateTransition translate =
    new TranslateTransition(Duration.millis(750));
translate.setToX(270);
translate.setToY(100);

ParallelTransition transition = new ParallelTransition(r,
translate);
transition.setCycleCount(Timeline.INDEFINITE);
transition.setAutoReverse(true);
transition.play();
```

A `ParallelTransition` implies that I can have multiple transitions applied to an object at any one point in time.

The transition below would make my square rotate. Notice that I have added it to the initialisation of transition so that it now applies translate and rotate.

```
RotateTransition rotate = new RotateTransition(Duration.millis(750));
rotate.setToAngle(360);

ParallelTransition transition = new ParallelTransition(r,
translate, rotate);
```

Here are two more useful transitions you can experiment with:

```
FillTransition fill = new FillTransition(Duration.millis(750));
fill.setToValue(Color.PEACHPUFF);

ScaleTransition scale = new ScaleTransition(Duration.millis(750));
scale.setToX(0.1);
scale.setToY(0.1);
```