**Intro to Drawing Graphics**

To draw some simple graphics, we first need to create a window. The easiest way to do this in the current version of Java is to create a JavaFX application. Previous versions of Java supported graphics through applets and Swing, but JavaFX is the current standard. When we create a JavaFX application we actually create a new window on the screen. Here is an example:

```
import javafx.application.Application;
import javafx.scene.canvas.Canvas;
import javafx.scene.Scene;
import javafx.scene.Group;
import javafx.stage.Stage;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.shape.ArcType;

public class HappyFace extends Application
{
   public static void main(String[] args)
   {
      launch(args);
   }

   @Override
   public void start(Stage primaryStage) throws Exception
   {
      Group root = new Group();
      Scene scene = new Scene(root);

      Canvas canvas = new Canvas(400, 300);
      GraphicsContext gc = canvas.getGraphicsContext2D();
      gc.strokeOval(100, 50, 200, 200);
      gc.fillOval(155, 100, 10, 20);
      gc.fillOval(230, 100, 10, 20);
      gc.strokeArc(150, 160, 100, 50, 180, 180, ArcType.OPEN);

      root.getChildren().add(canvas);
      primaryStage.setTitle("HappyFace in JavaFX");
      primaryStage.setScene(scene);
      primaryStage.show();
   }
}
```

When the program runs it displays:


The import section specifies a number of classes for Application, Canvas, Scene, Group, Stage, GraphicsContext, and ArcType. These are all components of JavaFX that are left for future discussion.

The line

```
public class HappyFace extends Application
```

begins the class definition for the program. It is named `HappyFace`. The words `extends Application` indicate that we are defining a JavaFX application, as opposed to some other kind of class. Although you need not worry about further details yet, we are using inheritance to create the class `HappyFace` based upon an existing class `Application`.

The application contains two methods – `main` and `start`. As you already know, the `main` method is where a Java program normally begins.

```
public static void main(String[] args)
{
   launch(args);
}
```

However, a JavaFX application is different. A JavaFX program begins execution in the `start` method. The `main` method is ignored in a correctly deployed JavaFX application. However, it is common to include `main` and a call to `launch` as a fallback, which will end up launching the JavaFX program and the `start` method.

For a JavaFX application, programs begin in the `start` method.

```
@Override
public void start(Stage primaryStage) throws Exception
```

For now, you can ignore the `@Override` and the `throws Exception` code. What you do need to know is that this method is invoked automatically when the JavaFX application is

run. JavaFX uses the metaphor of a stage and scenes, just like the stage and scene of a theater.

The next four lines sets up a canvas on a scene for you to draw simple graphics.

```
Group root = new Group();
Scene scene = new Scene(root);

Canvas canvas = new Canvas(400, 300);
GraphicsContext gc = canvas.getGraphicsContext2D();
```

At this point we can now use drawing operations on the canvas. The method invocation

```
gc.strokeOval(100, 50, 200, 200);
```

draws the big circle that forms the outline of the face. The first two numbers tell where on the screen the circle is drawn. The method `strokeOval`, as you may have guessed, draws ovals. The last two numbers give the width and height of the oval. To obtain a circle, you make the width and height the same size, as we have done here. The units for these numbers are called *pixels*, short for pixel element.

The two method invocations

```
gc.fillOval(155, 100, 10, 20);
gc.fillOval(230, 100, 10, 20);
```

draw the two eyes. The eyes are "real" ovals that are taller than they are wide. Also notice tht the method is called `fillOval`, not `strokeOval`, which means it draws an oval that is filled in.

The next invocation

```
gc.strokeArc(150, 160, 100, 50, 180, 180, ArcType.OPEN);
```

draws the mouth. We will explain the meaning of all these arguments in class, but it essentially specifies a bounding rectangle, start angle, stop angle, and type of arc.

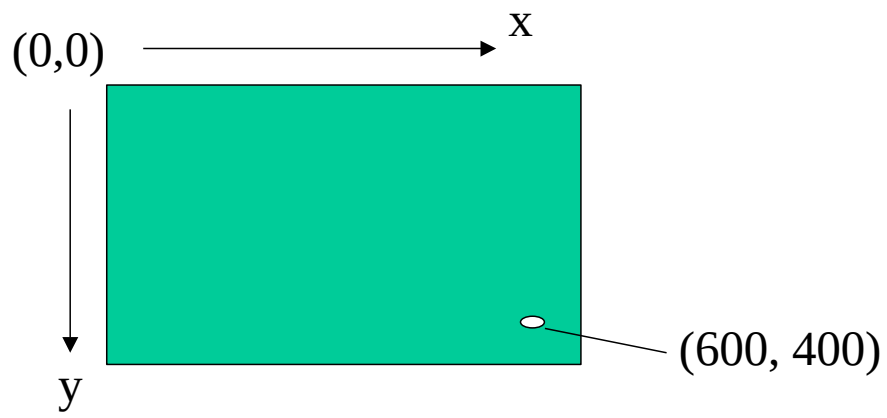
Finally, the block

```
root.getChildren().add(canvas);
primaryStage.setTitle("HappyFace in JavaFX");
primaryStage.setScene(scene);
primaryStage.show();
```

sets a title for the window and does some bookkeeping to set the stage and display the window.

## Drawing Coordinates

Once we have a graphics context to draw on, you can think of the screen as a grid of pixels where the upper left coordinate is 0,0. This is relative to where the canvas is on the viewframe. The x coordinate then grows out to the right, and the y coordinate grows down toward the bottom.

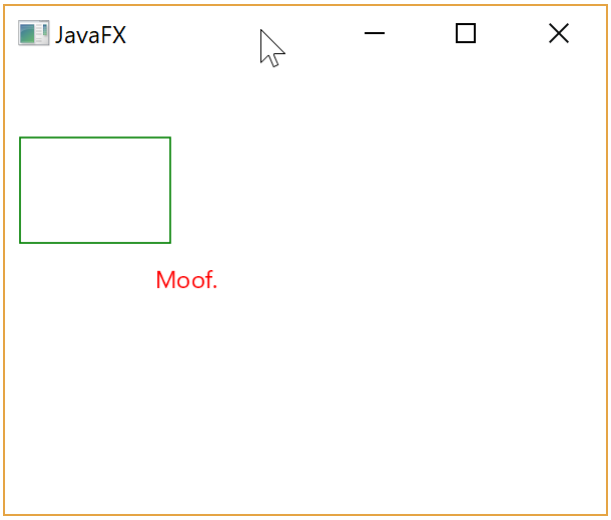



Here is some sample code that illustrates how to draw a rectangle and a string of text on the screen. You can use this as a template for your own programs, where you would likely want to draw something different inside the window in the start() method:

```
import javafx.application.Application;
import javafx.scene.canvas.Canvas;
import javafx.scene.Scene;
import javafx.scene.Group;
import javafx.stage.Stage;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;

public class Test extends Application
{
   public static void main(String[] args)
   {
      launch(args);
   }

   @Override
   public void start(Stage primaryStage) throws Exception
   {
      Group root = new Group();
      Scene scene = new Scene(root);
      Canvas canvas = new Canvas(400, 300);
      GraphicsContext gc = canvas.getGraphicsContext2D();

      // Set line / stroke color to Green
      gc.setStroke(Color.GREEN);
      // Draw rectangle at X=10, Y=50,
      // width=100, height = 70  pixels
      gc.strokeRect(10,50,100,70);
      // Write text at X = 100, Y = 150 in Red
      gc.setFill(Color.RED);
      gc.fillText("Moof.", 100, 150);

      root.getChildren().add(canvas);
      primaryStage.setTitle("JavaFX");
      primaryStage.setScene(scene);
```

```
        primaryStage.show();
    }
}
```

This code produces the following image:




`strokeRect(x1,y1,width,height)` draws a rectangle that is not filled in with the upper left corner at (x1,y1) and the specified width and height.

`fillText(string, x1, y1)` draws the text string at coordinates x1, y1.

**Colors**

Here are different colors available using the color class:

| | | |
|---|---|---|
| Color.BLACK | Color.DARKGRAY | Color.GRAY |
| Color.BLUE | Color.GREEN | Color.LIGHTGRAY |
| Color.CYAN | Color.MAGENTA | Color.ORANGE |
| Color.PINK | Color.RED | Color.WHITE |
| Color.YELLOW | | |

To create our own color, we can specify the color we want in RGB (red, green, blue) color model. This is an additive color model that is somewhat like mixing colors of light or like mixing paints. The idea is that any color is represented as some combination of red, green, and blue. To specify a color use:

```
Color.rgb(redvalue, greenvalue, bluevalue);
```

where each value is in the range 0-255.

For example:

```
Color.rgb(0,255,200);
```

Creates a green-blue color, with stronger green than blue. If red, green, and blue are all 0 then this is the color black. If red, green, and blue are all 255 then this is the color white. Red and green makes yellow, red and blue makes magenta, blue and green makes cyan, etc.

**More drawing functions**

Here is a list of other drawing functions you can use with the graphics context:

| |
|---|
| *gc.strokeOval*(X, Y, Width, Height)<br>Draws the outline of an oval having the specified width and height at the point (X, Y). |
| *gc.fillOval*(X, Y, Width, Height)<br>Same as strokeOval, but the oval is filled in. |
| *gc.strokeArc*(X, Y, Width, Height, StartAngle, ArcAngle, ArcType)<br>Draws an arc – that is, draws part of an oval. |
| *gc.fillArc*(X, Y, Width, Height, StartAngle, ArcAngle, ArcType)<br>Same as strokeArc, but the visible portion of the oval is filled in. |
| *gc.strokeRect*(X, Y, Width, Height)<br>Draws the outline of a rectangle having the specified width and height at the point (X, Y). |
| *gc.fillRect*(X, Y, Width, Height)<br>Same as strokeRect, but the rectangle is filled in. |
| *gc.strokeLine*(X1, Y1, X2, Y2)<br>Draws a line between points (X1, Y1) and (X2, Y2) |
| *gc.fillText*(String, X, Y)<br>Draws the specified string at point (X,Y) |
| *gc.setLineWidth*(Width)<br>Sets the current line width. |
| *gc.setFill*(Attribute)<br>Sets the current fill paint attribute. We have been using color as the fill attribute to set the current color, but gradients or image patterns are other attributes. |
| *gc.setFont*(Font)<br>Sets the current font. |

| |
|---|
| *gc.drawImage*(Image, X, Y)<br>Draws the specified image at point (X,Y) |
| *gc.setStroke*(Color)<br>Sets the stroke color. |
| *gc.setFill*(Color)<br>Sets the fill color. |

There are many more, for a full list see
https://docs.oracle.com/javase/8/javafx/api/javafx/scene/canvas/GraphicsContext.html

Here is an example using some of these functions to draw an ambivalent yellow face:

```
public void start(Stage primaryStage) throws Exception
{
   Group root = new Group();
   Scene scene = new Scene(root);
   Canvas canvas = new Canvas(400, 300);
   GraphicsContext gc = canvas.getGraphicsContext2D();

   gc.setFill(Color.YELLOW);
   gc.fillOval(50,50,200,200);
   gc.setFill(Color.rgb(0,0,255));     // Blue left eye
   gc.fillOval(100,100,20,10);
   gc.setFill(Color.rgb(0,255,255));   // Blue-Green right eye
   gc.fillOval(175,100,20,10);
   gc.setFill(Color.rgb(255,0,0));           // Red mouth
   gc.fillRect(115,200,50,4);

   root.getChildren().add(canvas);
   primaryStage.setTitle("David Bowie");
   primaryStage.setScene(scene);
   primaryStage.show();
}
```

This program draws the following image: