Chapter 14 JavaFX Basics

JavaFX vs Swing and AWT

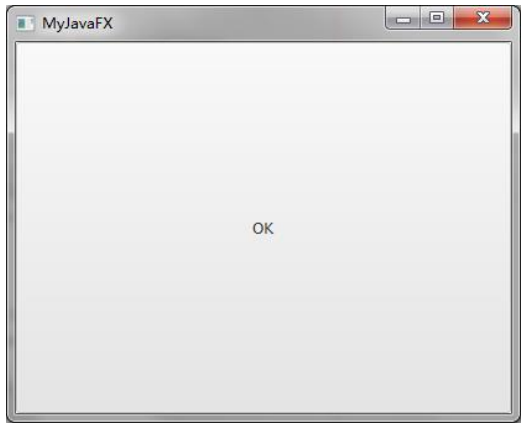
- 最早Java的GUI组件叫做AWT（Abstract Windows Toolkit)，后来发现AWT的设计有缺陷，同时比较耗费绘图资源，所以AWT已被废弃。
- 后来，Java重新设计了称为Swing的轻量级组件来替代AWT。Swing适合做桌面程序，然而随着互联网应用的兴起，桌面技术也慢慢过时了……
- 现在，Java设计了JavaFX，将RIA（Rich Internet Application）技术整合到桌面程序上，为两种程序界面提供了统一解决方案。也就是说，无论开发互联网应用程序还是桌面程序，界面代码是一样的，界面效果也是一样的。

第一个JavaFX程序

- 每一个JavaFX程序，都必须继承javafx.application.Application类。
- 先导入几个包，前3个是必须的，后一个看情况：

```
import javafx.application.Application; //程序框架
import javafx.stage.Stage; //舞台
import javafx.scene.Scene; //场景

import javafx.scene.control.Button; //做一个按钮
```

```
public class MyJavaFX extends Application {
  public void start(Stage primaryStage) {
    // 创建场景，并在上面放置一个按钮
    Button btOK = new Button("OK");
    Scene scene = new Scene(btOK, 400, 300);
    primaryStage.setTitle("MyJavaFX"); // 设置标题
    primaryStage.setScene(scene); // 把场景放上舞台
    primaryStage.show(); // 显示舞台
  }
  /* 现在main方法已经没有太多用处，写不写都行 */
  public static void main(String[] args) {
    Application.launch(args);
  }
}
```



解释

- 任何JavaFX程序都需要从Application类继承一个新类。
- main方法的存在是为了兼容一些过时的IDE，JavaFX不再把main做为程序入口，新入口是launch。
- launch方法用于启动JavaFX运行，这个方法会被JVM自动调用，所以不写launch也是可以的。launch会自动使用所在类的**无参构造方法**，构造出类的实例，并调用该类的start方法。所以，你的编程工作总是从start方法开始的。
- Java把整个窗口看成一个舞台Stage，为方便理解，我们看一个最最精简版的GUI程序。

舞台即是窗口

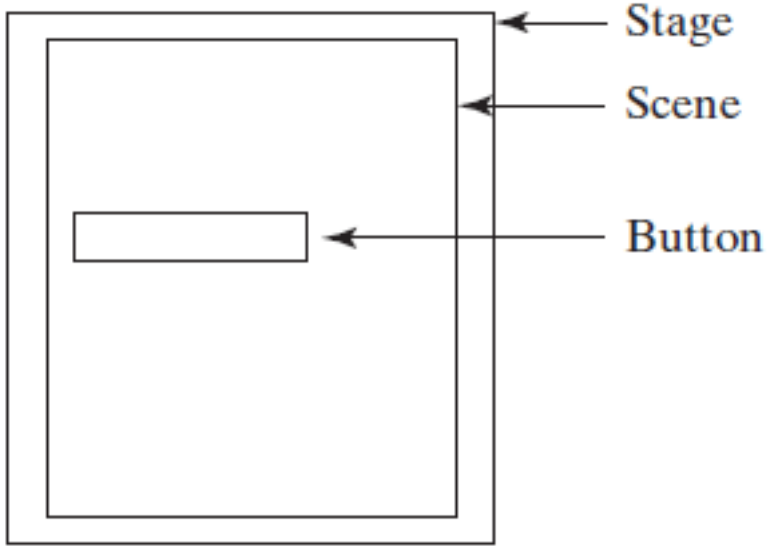
```
import javafx.application.Application;
import javafx.stage.Stage;
public class MyJavaFX extends Application
{
  public void start(Stage primaryStage) {
    primaryStage.show();
  }
}
```

- 默认情况下，舞台是可以用鼠标拖动改变大小的，如果你不想让用户捣乱，可以加上这句：

  primaryStage.setResizable(false);

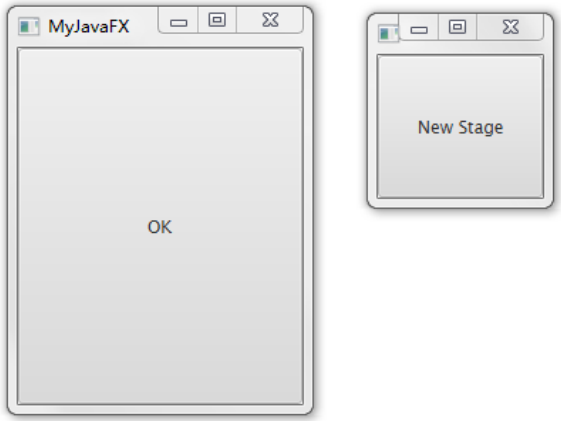
舞台，场景和按钮的关系

- 第一个例子中，界面层次是这样的：舞台是窗口，上面有一层场景，场景上有一个按钮。
- 场景比舞台小，因为它不包括标题栏。一个舞台上可以切换多个场景，这样界面就有了变化。
- 场景上可以摆放各种界面控件，例如按钮。
- 一个程序其实可以有多个舞台（就是多窗口）。



多舞台的例子

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
public class MultipleStageDemo extends Application {
  public void start(Stage primaryStage) {
      Scene scene = new Scene(new Button("OK"), 200, 250);
      primaryStage.setTitle("MyJavaFX");
      primaryStage.setScene(scene);
      primaryStage.show();
      Stage stage = new Stage();
      stage.setTitle("Second Stage");
      stage.setScene(new Scene(new Button("New Stage"), 100, 100));
      stage.show(); // Display the stage
    }
}
```



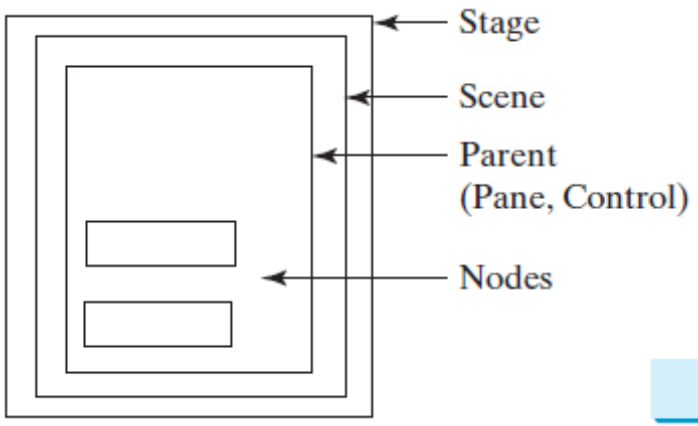
窗格，UI控件和形状
Panes, UI Controls, and Shapes

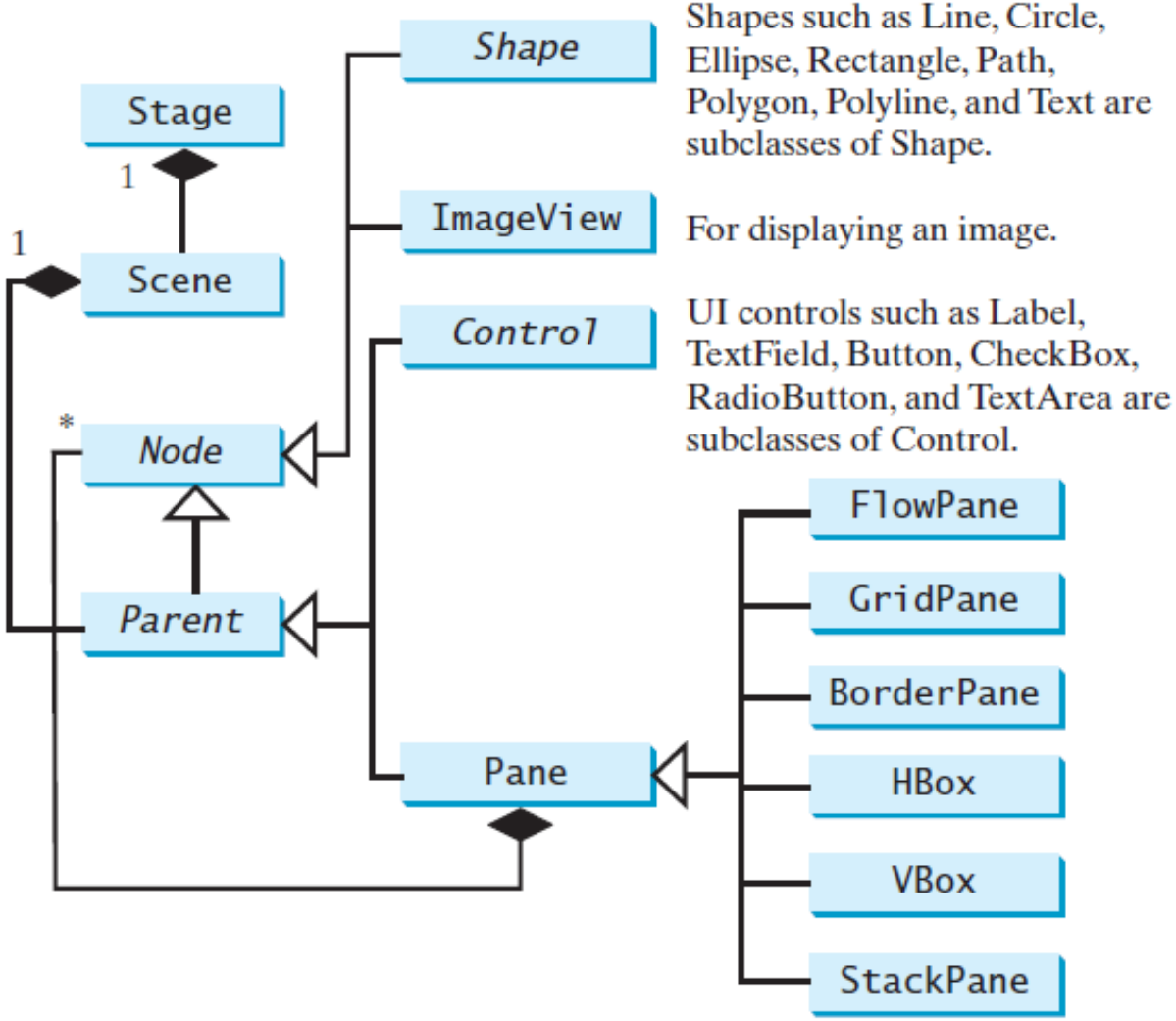
- 结点node指的是可视化的组件，例如一个形状，一个图片查看器，一个UI控件或者一个窗格pane。
- 形状指的是文本，线条，圆形，矩形等绘图元素。
- UI控件指的是标签，按钮，单选框，复选框，列表框，滚动条等界面控件。
- **除了场景，其它东西都不能直接放在舞台上显示**。它们只能随场景显示。

场景和窗格

- 场景上可以放置一个控件或者窗格，不过场景不能放置一个形状或者图片查看器。窗格是一种容器，它上面可以摆放任意结点。
- 场景的创建有两种方式：指定大小和默认大小
  - new Scene(Parent, width, height)
  - new Scene(Parent)
- 采用后者时，Java会自动计算显示场景所需要的大小。
- 其实，每一个结点都有一个无参构造方法，用于创建默认大小的结点。
- 一个标准GUI程序的界面层次如下图所示。

界面层次


Stage
Scene
Parent
(Pane, Control)
Nodes


Shape
Shapes such as Line, Circle, Ellipse, Rectangle, Path, Polygon, Polyline, and Text are subclasses of Shape.
Stage
1
1
Scene
ImageView
For displaying an image.
Control
UI controls such as Label, TextField, Button, CheckBox, RadioButton, and TextArea are subclasses of Control.
*
Node
Parent
FlowPane
GridPane
BorderPane
HBox
Pane
VBox
StackPane

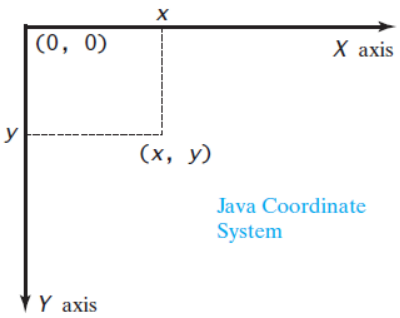
一个例子

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.stage.Stage;
5  import javafx.scene.layout.StackPane;
6
7  public class ButtonInPane extends Application {
8    @Override // Override the start method in the Application class
9    public void start(Stage primaryStage) {
10     // Create a scene and place a button in the scene
11     StackPane pane = new StackPane();
12     pane.getChildren().add(new Button("OK"));
13     Scene scene = new Scene(pane, 200, 50);
14     primaryStage.setTitle("Button in a pane"); // Set the stage title
15     primaryStage.setScene(scene); // Place the scene in the stage
16     primaryStage.show(); // Display the stage
17   }
18 }
```

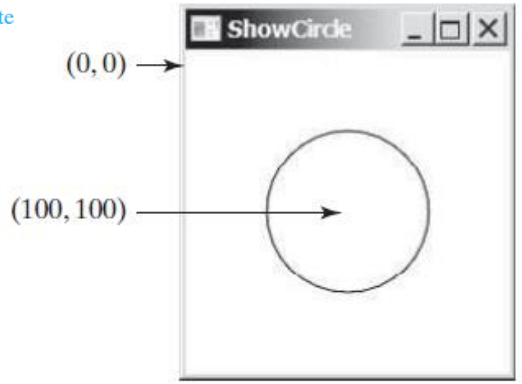
- 一个窗格可以贴很多个控件，所以用getChildren().add()

在窗格上画圆的例子（注意Java的坐标系统）

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.layout.Pane;
4  import javafx.scene.paint.Color;
5  import javafx.scene.shape.Circle;
6  import javafx.stage.Stage;
7
8  public class ShowCircle extends Application {
9    @Override // Override the start method in the Application class
10   public void start(Stage primaryStage) {
11     // Create a circle and set its properties
12     Circle circle = new Circle();
13     circle.setCenterX(100);
14     circle.setCenterY(100);
15     circle.setRadius(50);
16     circle.setStroke(Color.BLACK);
17     circle.setFill(Color.WHITE);
18
19     // Create a pane to hold the circle
20     Pane pane = new Pane();
21     pane.getChildren().add(circle);
22
23     // Create a scene and place it in the stage
24     Scene scene = new Scene(pane, 200, 200);
25     primaryStage.setTitle("ShowCircle"); // Set the stage title
26     primaryStage.setScene(scene); // Place the scene in the stage
27     primaryStage.show(); // Display the stage
28   }
29 }
```



Java Coordinate System



属性绑定Property Binding

- JavaFX引入了属性绑定的概念，可以让一个目标对象target object的值，始终跟随源对象source object的值而自动改变。
- 目标对象叫做绑定对象binding object或者绑定属性binding property，原对象叫做可绑定对象bindable object或者可观察对象 observable object。
- 简单点说，假如有两个属性d1和d2，如果d1绑定了d2，那每次d2的值改变后，d1的值也将自动改变。
- 注意必须是**属性**才有资格做绑定或被绑定，普通的数就不要拿来试了……

一个有趣的例子—死活要居中的圆

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;
public class ShowCircleCentered extends Application {
  public void start(Stage primaryStage) {
    Pane pane = new Pane();       Circle circle = new Circle();
    circle.centerXProperty().bind(pane.widthProperty().divide(2));
    circle.centerYProperty().bind(pane.heightProperty().divide(2));
    circle.setRadius(50);            circle.setStroke(Color.BLACK);
    circle.setFill(Color.WHITE);      pane.getChildren().add(circle);
    Scene scene = new Scene(pane, 200, 200);
    primaryStage.setTitle("ShowCircleCentered");
    primaryStage.setScene(scene);
    primaryStage.show();
  }
}
```

属性绑定的要求

- 属性绑定的语法很简单：target.bind(source);
- bind方法在 javafx.beans.property.Property接口中定义，所以绑定属性必须是 javafx.beans.property.Property的实例。源对象则必须是javafx.beans.value.ObservableValue接口的实例。
- 可绑定的值类型，可以是基本数据类型或字符串。对于double/float/long/int/boolean类型的值，绑定属性类型分别是DoubleProperty/FloatProperty/LongProperty/IntegerProperty/BooleanProperty。对字符串而言，绑定属性类型则是StringProperty。
- 这些属性都是ObservableValue的子类，所以它们可以做为源对象供绑定。

自定义绑定属性

- JavaFX中，每一个绑定属性（例如centerX），需要提供一个getter（例如 getCenterX())和一个setter（例如setCenterX(double))，而且还需要提供一个直接返回属性自己的getter（例如centerXProperty()）。下图中，左边是自定义绑定属性的模板，右边是示例。

```
public class SomeClassName {

  private PropertyType x;

  /** Value getter method */
  public propertyValueType getX() { ... }

  /** Value setter method */
  public void setX(propertyValueType value) { ... }

  /** Property getter method */
  public PropertyType
    xProperty() { ... }
}
```

```
public class Circle {

  private DoubleProperty centerX;

  /** Value getter method */
  public double getCenterX() { ... }

  /** Value setter method */
  public void setCenterX(double value) { ... }

  /** Property getter method */
  public DoubleProperty centerXProperty() { ... }
}
```

属性d1绑定属性d2的例子

```
1  import javafx.beans.property.DoubleProperty;
2  import javafx.beans.property.SimpleDoubleProperty;
3
4  public class BindingDemo {
5    public static void main(String[] args) {
6      DoubleProperty d1 = new SimpleDoubleProperty(1);
7      DoubleProperty d2 = new SimpleDoubleProperty(2);
8      d1.bind(d2);
9      System.out.println("d1 is " + d1.getValue()
10       + " and d2 is " + d2.getValue());
11     d2.setValue(70.2);
12     System.out.println("d1 is " + d1.getValue()
13       + " and d2 is " + d2.getValue());
14   }
15 }
```

```
d1 is 2.0 and d2 is 2.0
d1 is 70.2 and d2 is 70.2
```

解释一下上面的例子

1. 我们使用了new SimpleDoubleProperty(1)，原因是DoubleProperty, FloatProperty,LongProperty, IntegerProperty, BooleanProperty都是抽象类，仅作为类型使用；它们的子类SimpleDoubleProperty, SimpleFloatProperty, SimpleLongProperty, SimpleIntegerProperty, SimpleBooleanProperty才是可以直接new出来的具体类。
2. bind方法实现的是属性d1对d2的**单向绑定**（unidirectional binding），也就是d2能够影响d1，反之不行；如果需要两种属性互相影响，可以用bindBidirectional方法，实现属性d1和d2的**双向绑定**。

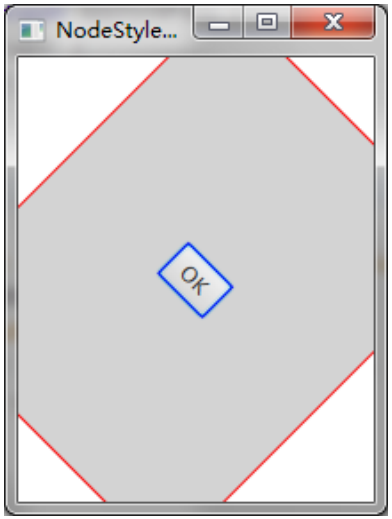
结点的常见属性和方法

- 结点Node是一个抽象类，定义了许多属性和方法。其中的两个是style和rotate。
- 网页上的HTML采用层叠样式表cascading style sheets (CSS)来定义外观，JavaFX将其思想引入了界面设计，采用JavaFX CSS来定义，用于生成和网页效果一致的界面。
- 为避免混淆，JavaFX CSS和网页的CSS并不兼容，前者的所有样式名称都有一个前缀**-fx-**。语法上，两者是一致的，采用styleName:value的形式，多个样式之间用分号(;)作为分隔符。

setStyle和setRotate

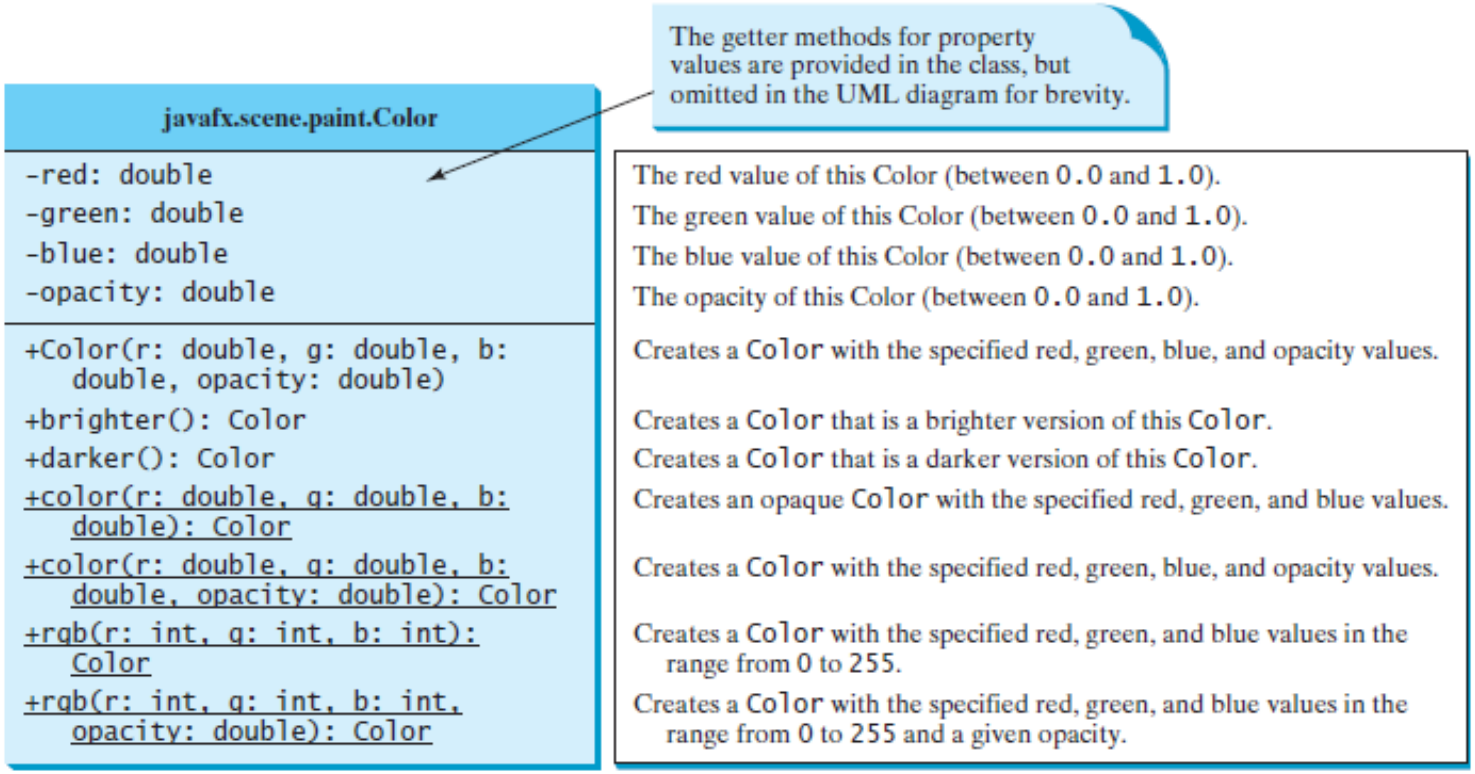
- 利用JavaFX CSS可以很方便设置结点外观，例如：
  - circle.setStyle("-fx-stroke: black; -fx-fill: red;");
- 以前为了实现上述效果，需要两个语句：
  - circle.setStroke(Color.BLACK);
  - circle.setFill(Color.RED);
- 如果你写错了JavaFX样式表，程序并不报错，只是忽略这个样式设置而已。
- 旋转结点可以采用setRotate，例如：
  - button.setRotate(**8o**);
  - 注意旋转中心是结点中心，旋转单位是角度，并且旋转的正方向是顺时针。

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.layout.StackPane;
public class NodeStyleRotateDemo extends Application {
  public void start(Stage primaryStage) {
    StackPane pane = new StackPane();
    Button btOK = new Button("OK");
    btOK.setStyle("-fx-border-color: blue;");
    pane.getChildren().add(btOK);
    pane.setRotate(45);
    pane.setStyle("-fx-border-color: red; -fx-background-color:
lightgray;");
    Scene scene = new Scene(pane, 200, 250);
    primaryStage.setTitle("NodeStyleRotateDemo");
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
}
```



Color类

Java用4个数字表示颜色。前3个数字分别表示red, green, blue分量的强度，0.0或0表示没有，1.0或255表示最强。除此之外，颜色还有一个透明度属性，取值0.0~1.0。注意透明度不能取整数。



Color类的例子

颜色的透明度又称为alpha，所以Java的颜色模型就是俗称的RGBA模型。透明度0.0表示完全透明，1.0表示完全不透明。

下面是几个例子：

```
Color red = new Color(255, 0, 0);
Color black = new Color(0, 0, 0);
Color white = new Color(255, 255, 255);
Color color = new Color(0.25, 0.14, 0.333,
 0.51);
```

注意颜色一旦创建就是不可变的，例如想单独修改red分量，这是做不到的。

颜色常量

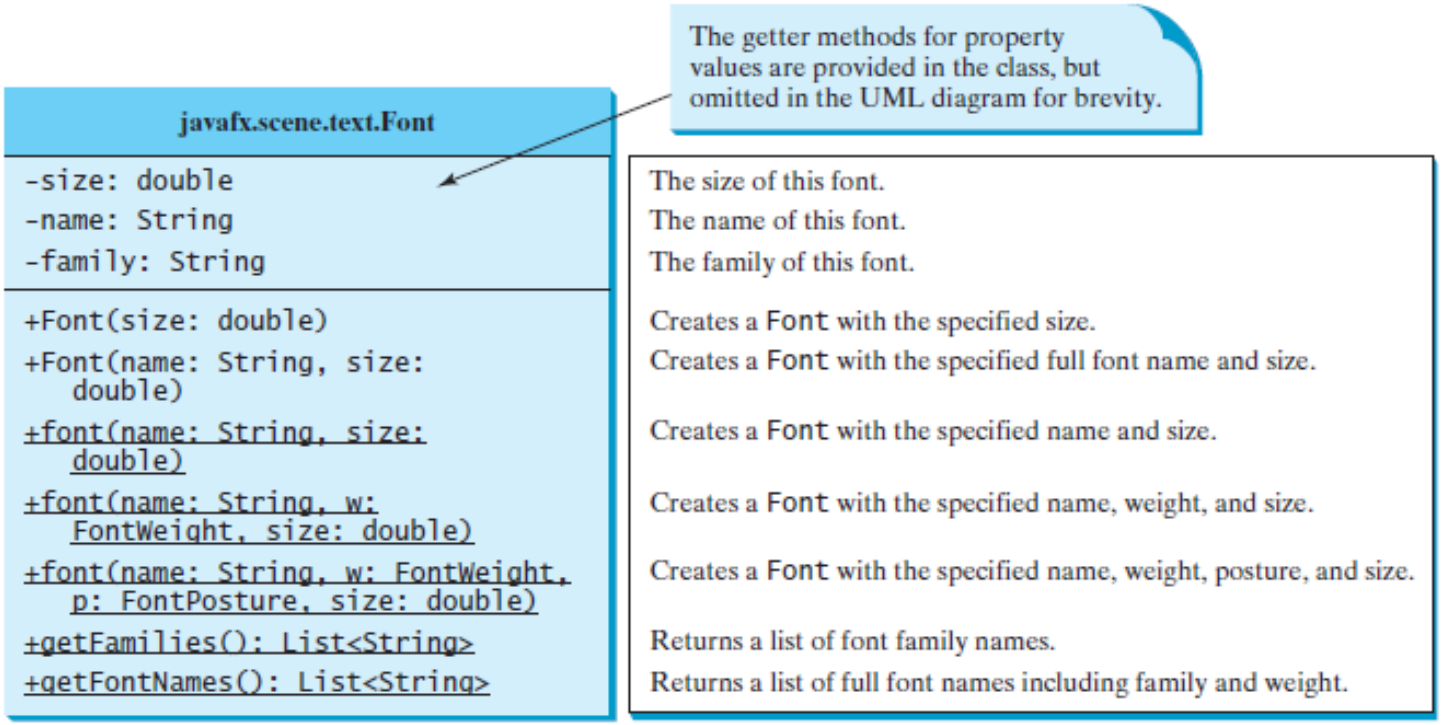
- 常见的几个颜色被预定义为Color中的常量，它们是BEIGE, BLACK, BLUE,BROWN, CYAN, DARKGRAY, GOLD, GRAY, GREEN, LIGHTGRAY, MAGENTA, NAVY, ORANGE, PINK,RED, SILVER, WHITE, YELLOW

- 例如需要把circle填充为红色，可以直接用：

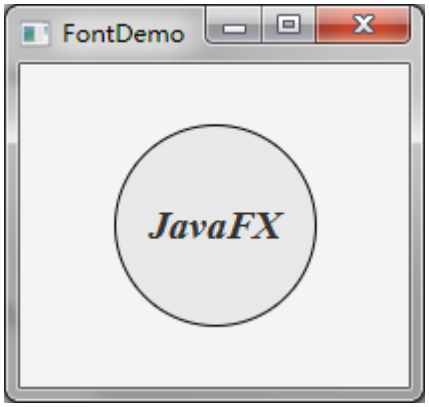
```
circle.setFill(Color.RED);
```

字体类Font

- 字体类的使用是很简单的，例如：
  - Font font1 = **new** Font("**SansSerif**", **16**);
  - Font font2 = Font.font("**Times New Roman**", FontWeight.BOLD, FontPosture.ITALIC, **12**);



```
//此处省略若干import
public class FontDemo extends Application {
  public void start(Stage primaryStage) {
    Pane pane = new StackPane();
    Circle circle = new Circle();
    circle.setRadius(50);
    circle.setStroke(Color.BLACK);
    circle.setFill(new Color(0.5, 0.5, 0.5, 0.1));
    pane.getChildren().add(circle);
    Label label = new Label("JavaFX");
    label.setFont(Font.font("Times New Roman",
        FontWeight.BOLD, FontPosture.ITALIC, 20));
    pane.getChildren().add(label);
    Scene scene = new Scene(pane);
    primaryStage.setTitle("FontDemo");
    primaryStage.setScene(scene);
    primaryStage.show();
  }
}
```



Image类和ImageView类

- Image类可以从本地或者网络上读取指定图像，例如：
  - new Image("image/us.gif")
  - new Image("http://www.cs.armstrong.edu/liang/image/us.gif")
  - 目前仅支持四种类型图片：BMP, GIF, JPEG, PNG
- ImageView用来显示图像，可配合Image使用，例如：
  - Image image = **new** Image("**image/us.gif**");
  - ImageView imageView = **new** ImageView(image);
- 或者单独使用，例如：
  - ImageView imageView = **new** ImageView("**image/us.gif**");

显示图片的例子

```
10  public class ShowImage extends Application {
11    @Override // Override the start method in the Application class
12    public void start(Stage primaryStage) {
13      // Create a pane to hold the image views
14      Pane pane = new HBox(10);
15      pane.setPadding(new Insets(5, 5, 5, 5));
16      Image image = new Image("image/us.gif");
17      pane.getChildren().add(new ImageView(image));
18
19      ImageView imageView2 = new ImageView(image);
20      imageView2.setFitHeight(100);
21      imageView2.setFitWidth(100);
22      pane.getChildren().add(imageView2);
23
24      ImageView imageView3 = new ImageView(image);
25      imageView3.setRotate(90);
26      pane.getChildren().add(imageView3);
27
28      // Create a scene and place it in the stage
29      Scene scene = new Scene(pane);
30      primaryStage.setTitle("ShowImage"); // Set the stage title
31      primaryStage.setScene(scene); // Place the scene in the stage
32      primaryStage.show(); // Display the stage
33    }
34  }
```

布局窗格（Layout Panes）

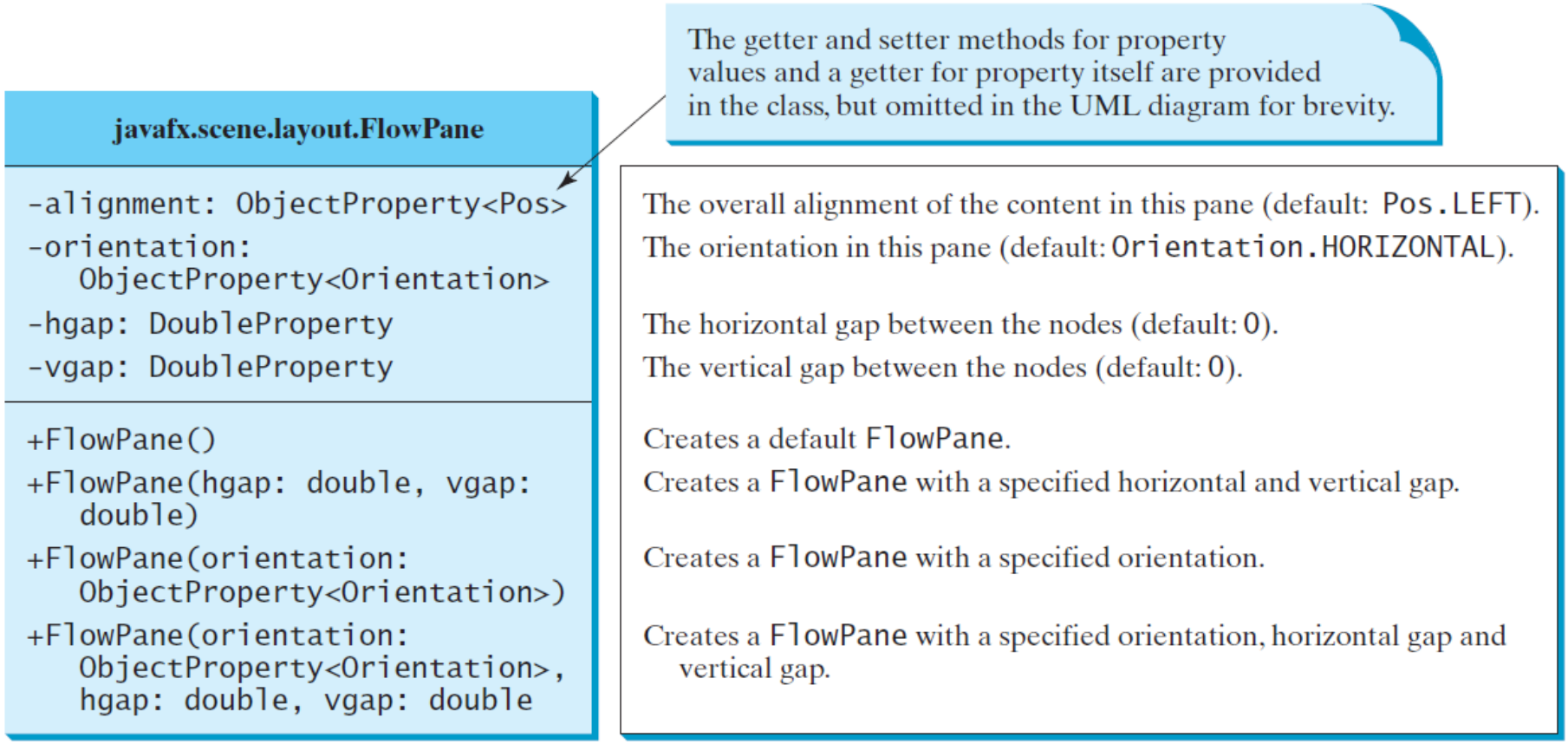
- 布局窗格是一种容器，用来摆放各种结点。JavaFX提供了多种窗格，用于支持不同的摆放方式。其中有几个我们曾经见过，Pane, StackPane, Hbox，亲，还记得不？

Class	*Description*
`Pane`	Base class for layout panes. It contains the `getChildren()` method for returning a list of nodes in the pane.
`StackPane`	Places the nodes on top of each other in the center of the pane.
`FlowPane`	Places the nodes row-by-row horizontally or column-by-column vertically.
`GridPane`	Places the nodes in the cells in a two-dimensional grid.
`BorderPane`	Places the nodes in the top, right, bottom, left, and center regions.
`HBox`	Places the nodes in a single row.
`VBox`	Places the nodes in a single column.

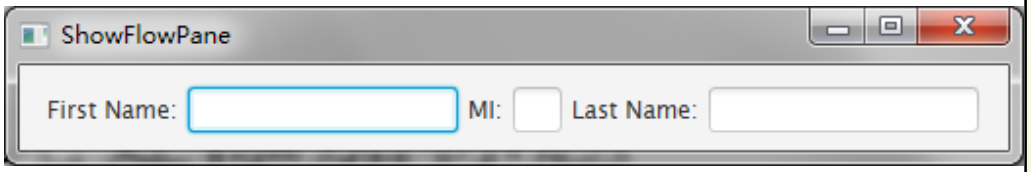
流式窗格FlowPane

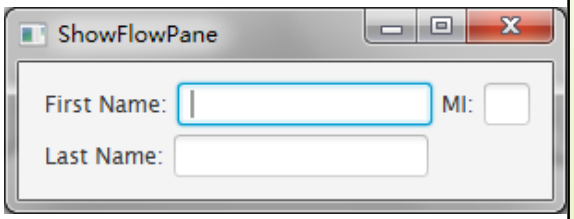
- 流式窗格用于水平从左到右，或者垂直从上到下摆放结点。两种摆放方式通过Orientation.HORIZONTAL 或者Orientation.VERTICAL指定。
- 流式窗格有几个绑定属性alignment, orientation, hgap, vgap，表示对齐，朝向，水平间距，垂直间距，你可以利用属性绑定技术，让结点在舞台大小改变的时候，依然保持在合适的位置。其实这几个属性在其它类型的窗格中也会出现。

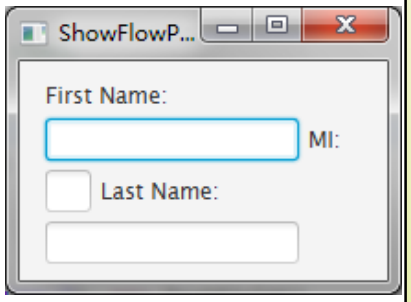
FlowPane的UML图



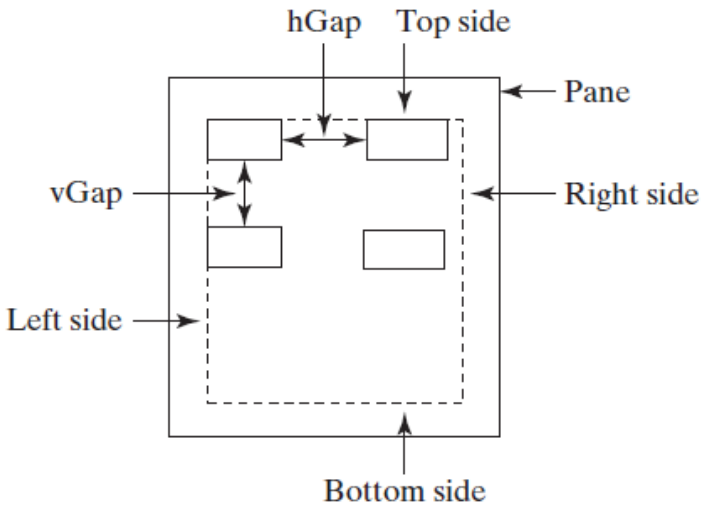
```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;
public class Hello extends Application {
  @Override
  public void start(Stage primaryStage) {
    FlowPane pane = new FlowPane();
    pane.setPadding(new Insets(11, 12, 13, 14));
    pane.setHgap(5);
    pane.setVgap(5);
    pane.getChildren().addAll(new Label("First Name:"), new TextField(),
new Label("MI:"));
    TextField tfMi = new TextField();
    tfMi.setPrefColumnCount(1);
    pane.getChildren().addAll(tfMi, new Label("Last Name:"), new TextField());
    Scene scene = new Scene(pane, 200, 250);
    primaryStage.setTitle("ShowFlowPane"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
}
```







程序解读—1



- FlowPane可以利用Insets类定义边距。
- 四个边距可以按照顺时针方向分别给出，例如Insets(11, 12, 13, 14)，表示上(11), 右 (12), 下(13), 左(14)，单位是像素。
- 如果四个边距相等，可以简单用：Insets(value)
- hGap和vGap属性用于指定结点之间的水平间距和垂直间距。

程序解读—2

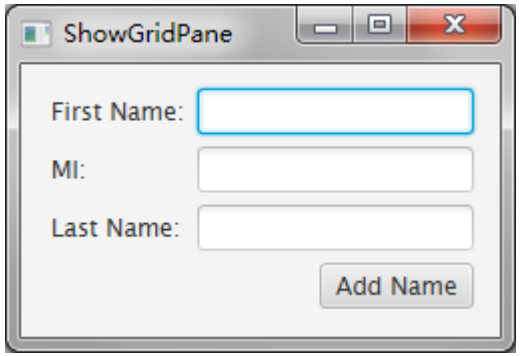
- 每一个FlowPane都有一个结点列表ObservableList，用于存储贴在它上面的结点。这个列表可以通过getChildren()方法返回。
- 往FlowPane上面添加结点，可以有两种方式：
  - add(node) 或 addAll(node1, node2, …)
- 同样，也可以从FlowPane 删除结点，可以有两种方式删除：
  - remove(node) 或 removeAll()
- tfMi.setPrefColumnCount(1) 用于指定文本框的显示列宽，这里是1，相当于正好能够显示一个字符。注意这招并不能限制用户输入的长度。

网格窗格GridPane

- 网格窗格是把整个窗格平均划分为m行n列，网格的编号和矩阵下标编号方式一致，左上角(0,0)，第一个分量表示行，第二个分量表示列。
- 用户可以将结点放置在指定的(i,j)处。
- 看个例题，我们先导入几个包：

```
import javafx.application.Application;
import javafx.geometry.*;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
```

```
public class ShowGridPane extends Application {
  public void start(Stage primaryStage) {
    GridPane pane = new GridPane();
    pane.setAlignment(Pos.CENTER);
    pane.setPadding(new Insets(11.5, 12.5, 13.5, 14.5));
    pane.setHgap(5.5);
    pane.setVgap(5.5);
    pane.add(new Label("First Name:"), 0, 0);
    pane.add(new TextField(), 1, 0);
    pane.add(new Label("MI:"), 0, 1);
    pane.add(new TextField(), 1, 1);
    pane.add(new Label("Last Name:"), 0, 2);
    pane.add(new TextField(), 1, 2);
    Button btAdd = new Button("Add Name");
    pane.add(btAdd, 1, 3);
    GridPane.setHalignment(btAdd, HPos.RIGHT);
    Scene scene = new Scene(pane);
    primaryStage.setTitle("ShowGridPane");
    primaryStage.setScene(scene);
    primaryStage.show();
  }
}
```



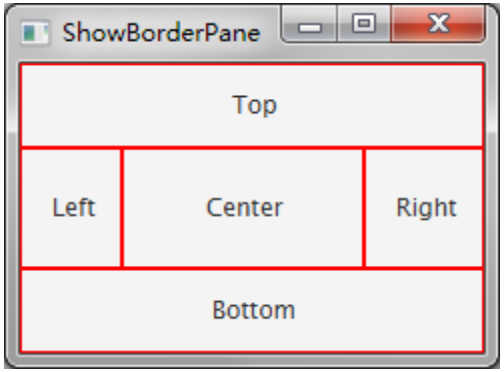
程序解读

- setAlignment(Pos.CENTER);用于指定单元格默认对齐方式（水平、垂直居中）；
- 添加结点可以用add(Node, columnIndex, rowIndex)的方式，删除结点可以用getChildren().remove(node)或者getChildren().removeAll()
- 程序不需要指定网格划分的行列数m和n，JavaFX会根据所有结点的位置信息自动计算需要的行列数。
- setHalignment(btAdd, HPos.RIGHT);用于对按钮单独设置水平右对齐。

边界窗格BorderPane

- 边界窗格是将窗格划分为5个区域，分别为上下左右中，然后将结点利用setTop(node), setBottom(node), setLeft(node), setRight(node), setCenter(node) 方法添加到指定区域中。
- 为了标注这5个区域给大家欣赏一下，我们需要在这5个区域上写字，因此自定义了一个可以显示指定文字的窗格CustomPane。

```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class ShowBorderPane extends Application {
  public void start(Stage primaryStage) {
   BorderPane pane = new BorderPane();
   pane.setTop(new CustomPane("Top")); pane.setRight(new CustomPane("Right"));
   pane.setBottom(new CustomPane("Bottom")); pane.setLeft(new CustomPane("Left"));
   pane.setCenter(new CustomPane("Center"));
   Scene scene = new Scene(pane);
   primaryStage.setTitle("ShowBorderPane"); // Set the stage title
   primaryStage.setScene(scene); // Place the scene in the stage
   primaryStage.show(); // Display the stage
  }
}
class CustomPane extends StackPane {
  public CustomPane(String title) {
   getChildren().add(new Label(title)); setStyle("-fx-border-color: red");
   setPadding(new Insets(11.5, 12.5, 13.5, 14.5));
  }
}
```

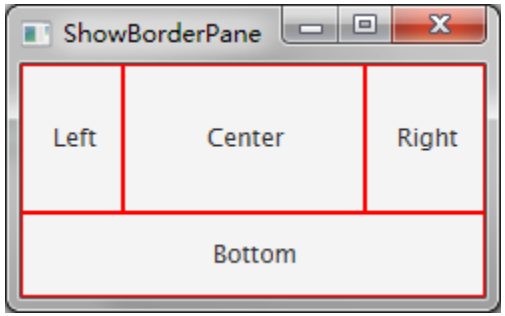


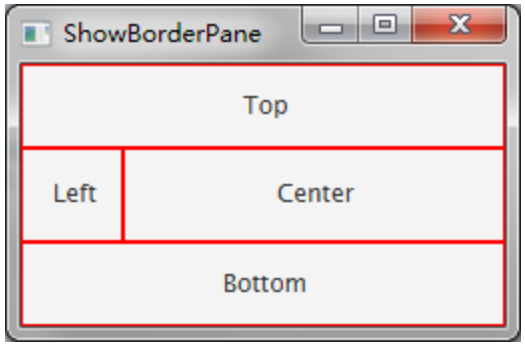
程序解读

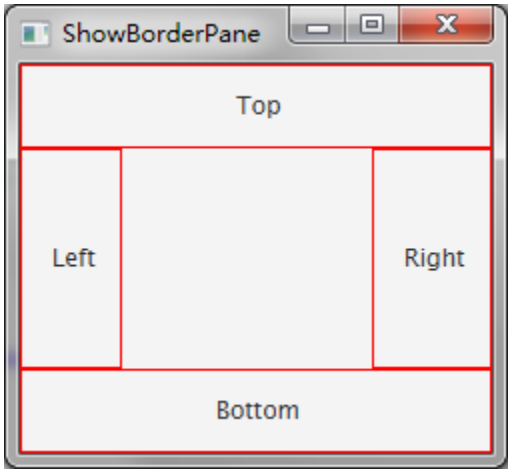
- 这个例子告诉我们，一个窗格可以当作一个结点，并被放置到另一个窗格上，本题中，一个CustomPane（本质上是StackPane）被放置在BorderPane上。
- 利用这种窗格套窗格的技术，可以制作出很复杂的界面。
- 边界窗格的每一个位置只能放置一个结点，所以不存在类似add或者getChildren的方法，为了删除结点，可以将一个空结点替代原结点，例如setTop(null)。
- 如果一个位置没有结点，该位置不一定会被留空，也可能被完全挤占。例如下面的示意：

因为Center会自动留空，建议程序优先使用Center区域。

- 没有Top、没有Right和没有Center的示意图







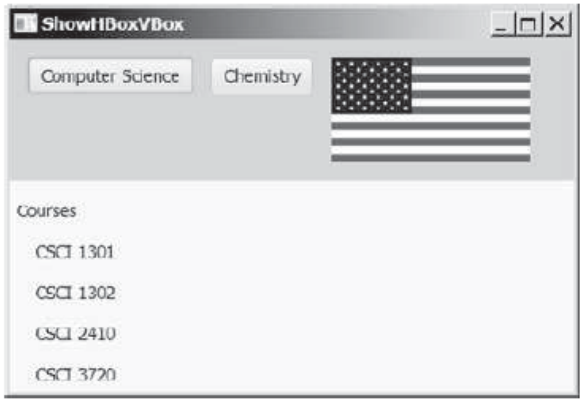
水平盒与垂直盒
HBox and VBox

- HBox用于将所有结点排成一行，与FlowPane不同，即使宽度不够，HBox也不会将结点换行。与此类似，VBox将所有结点排成一列。
- 先导入一堆包，看一个例子：

```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.stage.Stage;
import javafx.scene.image.*;
```

```
public class ShowHBoxVBox extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
  // Create a border pane
  BorderPane pane = new BorderPane();
  // Place nodes in the pane
  pane.setTop(getHBox());
  pane.setLeft(getVBox());
  // Create a scene and place it in the stage
  Scene scene = new Scene(pane);
  primaryStage.setTitle("ShowHBoxVBox"); // Set the stage title
  primaryStage.setScene(scene); // Place the scene in the stage
  primaryStage.show(); // Display the stage
 }
```

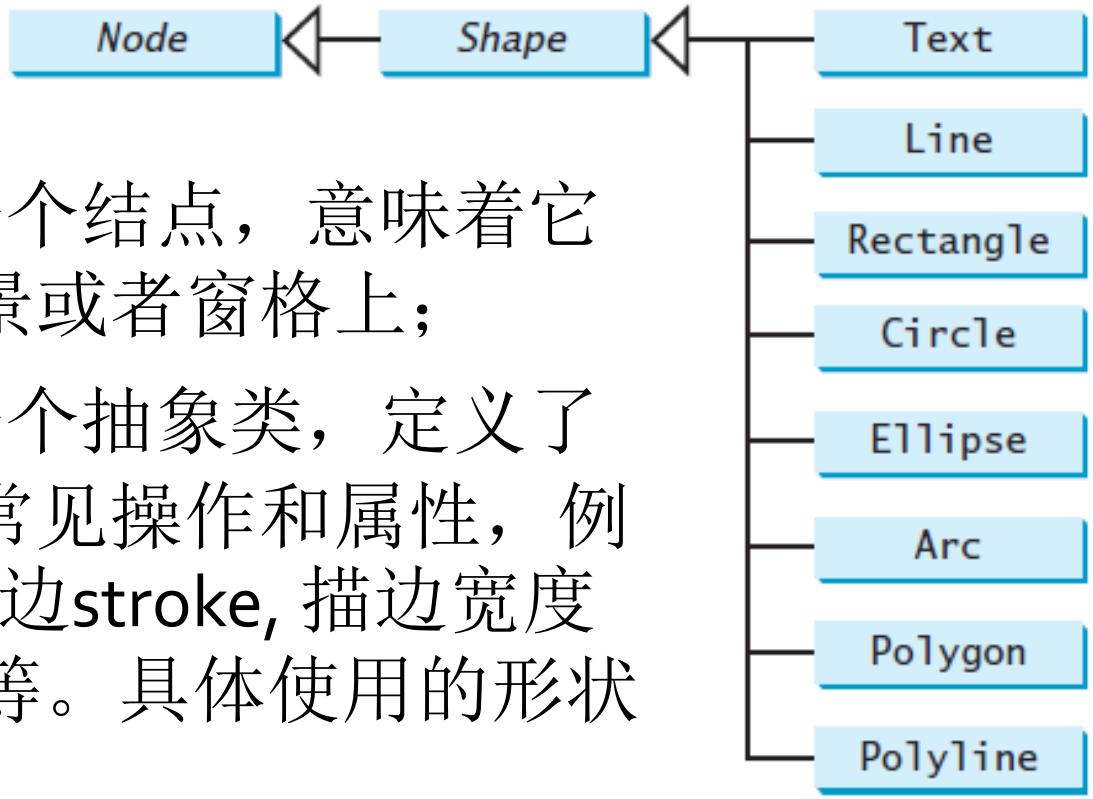
```
private HBox getHBox() {
  HBox hBox = new HBox(15);
  hBox.setPadding(new Insets(15, 15, 15, 15));
  hBox.setStyle("-fx-background-color: gold");
  hBox.getChildren().add(new Button("Computer Science"));
  hBox.getChildren().add(new Button("Chemistry"));
  ImageView imageView = new ImageView(new Image("image/us.gif"));
  hBox.getChildren().add(imageView);
  return hBox;
}
private VBox getVBox() {
  VBox vBox = new VBox(15);
  vBox.setPadding(new Insets(15, 5, 5, 5));
  vBox.getChildren().add(new Label("Courses"));
  Label[] courses = {new Label("CSCI 1301"), new Label("CSCI 1302"),
      new Label("CSCI 2410"), new Label("CSCI 3720")};
  for (Label course: courses) {
    VBox.setMargin(course, new Insets(0, 0, 0, 15));
    vBox.getChildren().add(course);
  }
  return vBox;
}
}
```



程序解读

- 程序定义了getHBox()方法用于返回一个HBox对象，上面包含了2个按钮和1个图像查看器。HBox的背景色采用Java CSS设置。
- getVBox()方法用于返回一个VBox对象，其中包含5个标签。 setMargin方法用来指定VBox中结点的边距。
- 利用单独的方法一个个制作复杂的界面元素，最后统一放置到主界面中，符合模块化的编程思想，推荐使用。

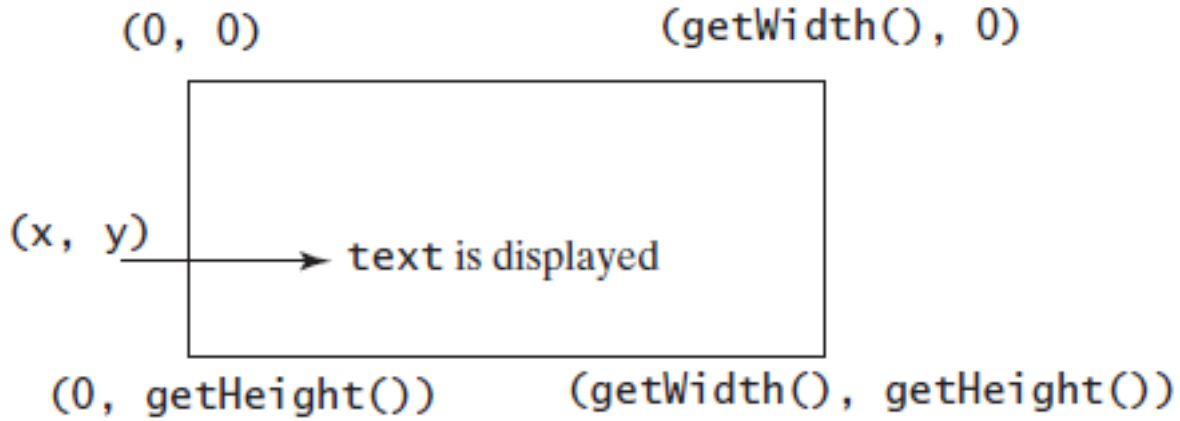
形状Shape



- Shape类是一个结点，意味着它可以贴在场景或者窗格上；
- Shape类是一个抽象类，定义了几何形状的常见操作和属性，例如填充fill, 描边stroke, 描边宽度strokeWidth等。具体使用的形状如右图：

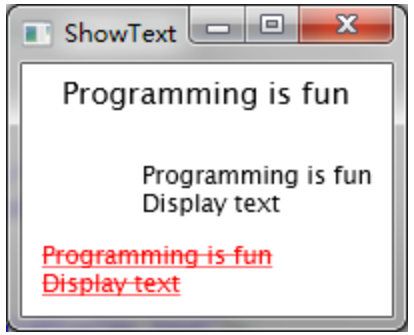
文本Text

- 文本是按照指定位置和宽度、高度显示的。具体显示约定如下（水平方向为x，垂直方向为y）：



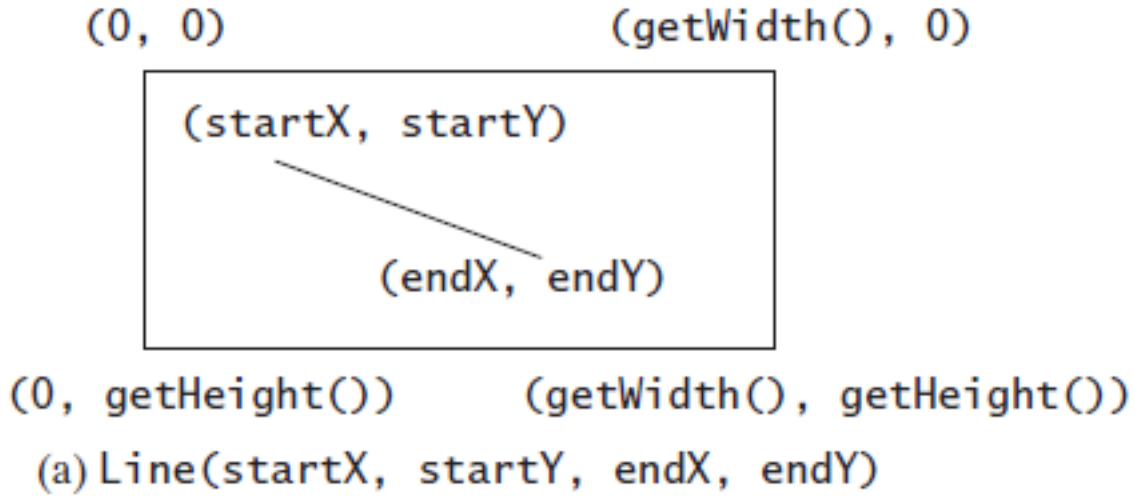
- 文本的构造方法指定的是文本显示的左下角(x,y)，显示宽度和高度是由文本的字体决定的。

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.geometry.Insets;
import javafx.stage.Stage;
import javafx.scene.text.*;
public class ShowText extends Application {
  public void start(Stage primaryStage) {
    Pane pane = new Pane();
    pane.setPadding(new Insets(5, 5, 5, 5));
    Text text1 = new Text(20, 20, "Programming is fun");
    text1.setFont(Font.font("Courier", FontWeight.BOLD, FontPosture.ITALIC, 15));
    pane.getChildren().add(text1);
    Text text2 = new Text(60, 60, "Programming is fun\nDisplay text");
    pane.getChildren().add(text2);
    Text text3 = new Text(10, 100, "Programming is fun\nDisplay text");
    text3.setFill(Color.RED);
    text3.setUnderline(true);
    text3.setStrikethrough(true);
    pane.getChildren().add(text3);
    Scene scene = new Scene(pane);
    primaryStage.setTitle("ShowText"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
}
```



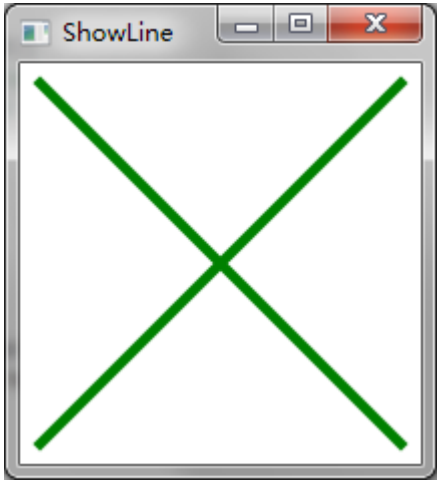
线Line

- 线是由两点决定的，所以需要四个参数：startX, startY, endX, endY，如图：



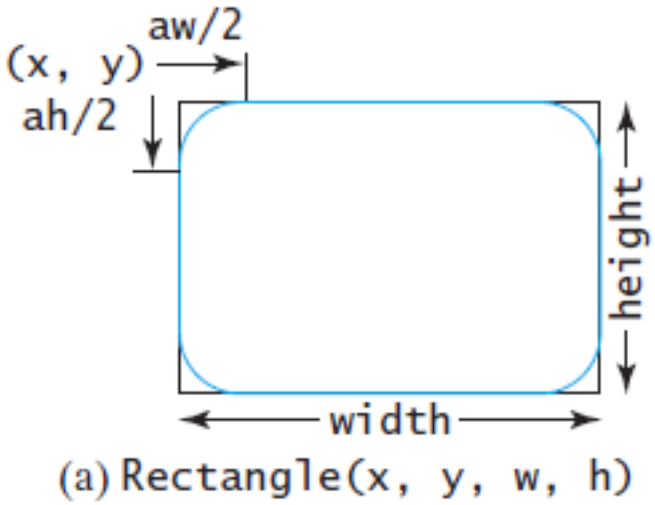
(a) Line(startX, startY, endX, endY)

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.scene.shape.Line;
public class ShowLine extends Application {
public void start(Stage primaryStage) {
    Scene scene = new Scene(new LinePane(), 200, 200);
    primaryStage.setTitle("ShowLine"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
}
class LinePane extends Pane {
  public LinePane() {
    Line line1 = new Line(10, 10, 10, 10);
    line1.endXProperty().bind(widthProperty().subtract(10));
    line1.endYProperty().bind(heightProperty().subtract(10));
    line1.setStrokeWidth(5); line1.setStroke(Color.GREEN);
    getChildren().add(line1);
    Line line2 = new Line(10, 10, 10, 10);
    line2.startXProperty().bind(widthProperty().subtract(10));
    line2.endYProperty().bind(heightProperty().subtract(10));
    line2.setStrokeWidth(5); line2.setStroke(Color.GREEN);
    getChildren().add(line2);
  }
}
```



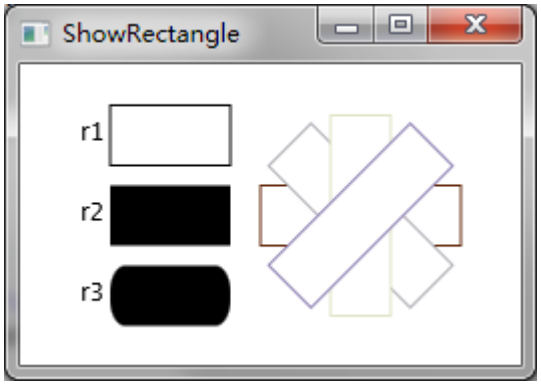
矩形Rectangle

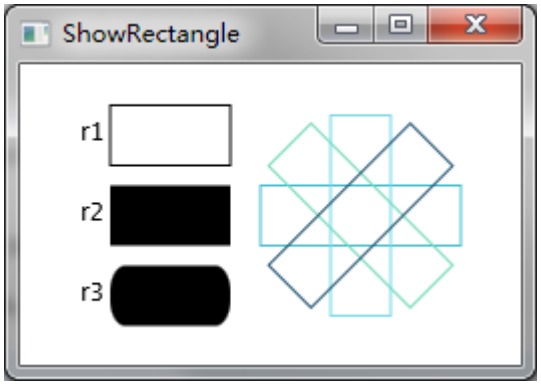
- 矩形由四个参数确定，左上角坐标x, y, 宽度width, 高度height。
- 矩形还有一个圆角属性，不过只能通过下面两个方法设置。
  - setArcWidth
  - setArcHeight
- 圆角的效果，是由aw和ah构成的椭圆做成的，所以图上才会标注aw/2和ah/2。



(a) Rectangle(x, y, w, h)

```
13      Pane pane = new Pane();
14
15      // Create rectangles and add to pane
16      Rectangle r1 = new Rectangle(25, 10, 60, 30);
17      r1.setStroke(Color.BLACK);
18      r1.setFill(Color.WHITE);
19      pane.getChildren().add(new Text(10, 27, "r1"));
20      pane.getChildren().add(r1);
21
22      Rectangle r2 = new Rectangle(25, 50, 60, 30);
23      pane.getChildren().add(new Text(10, 67, "r2"));
24      pane.getChildren().add(r2);
25
26      Rectangle r3 = new Rectangle(25, 90, 60, 30);
27      r3.setArcWidth(15);
28      r3.setArcHeight(25);
29      pane.getChildren().add(new Text(10, 107, "r3"));
30      pane.getChildren().add(r3);
31
32      for (int i = 0; i < 4; i++) {
33        Rectangle r = new Rectangle(100, 50, 100, 30);
34        r.setRotate(i * 360 / 8);
35        r.setStroke(Color.color(Math.random(), Math.random(),
36            Math.random()));
37        r.setFill(Color.WHITE);
38        pane.getChildren().add(r);
39      }
```

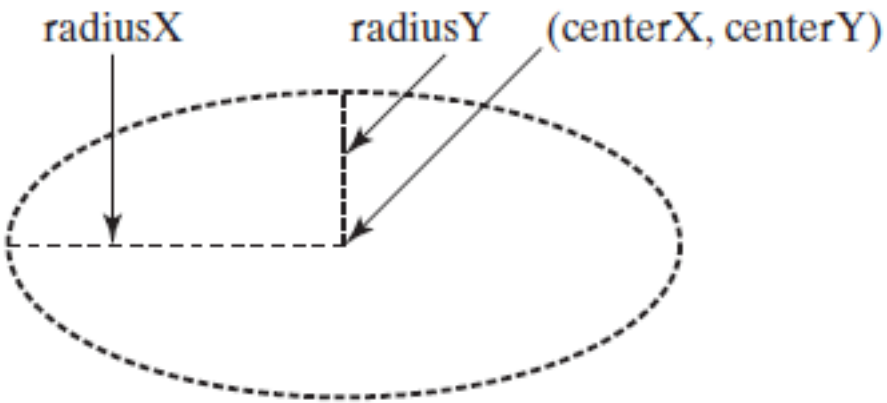




改成r.setFill(null);
的效果就是下图

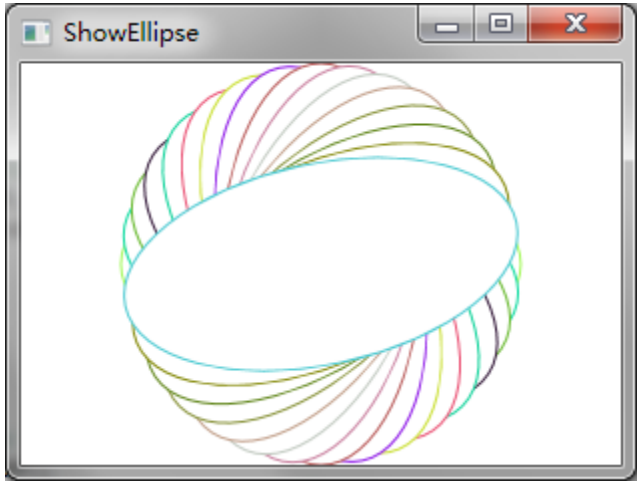
圆和椭圆**Circle** and **Ellipse**

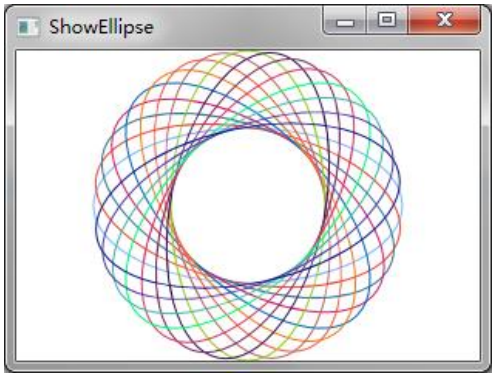
- 圆有三个参数centerX, centerY, radius，椭圆有四个参数centerX, centerY, radiusX, radiusY。因为椭圆比较复杂，我们就看这个：



(a) `Ellipse(centerX, centerY, radiusX, radiusY)`

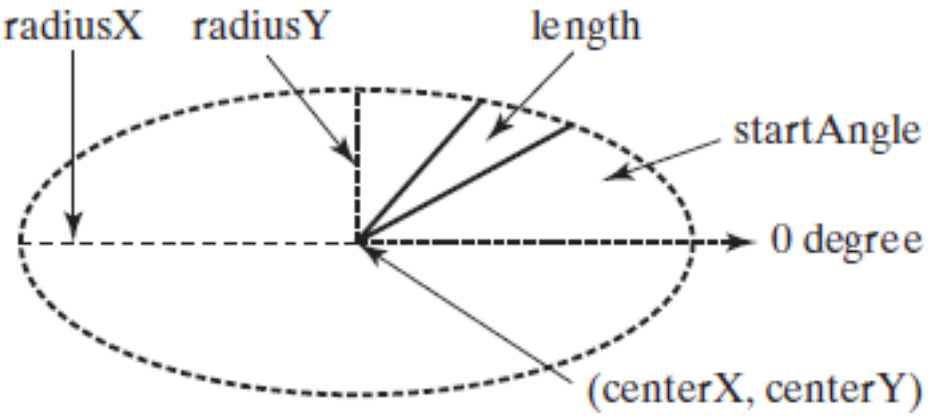
```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.scene.shape.Ellipse;
public class ShowEllipse extends Application {
  public void start(Stage primaryStage) {
    Pane pane = new Pane();
    for (int i = 0; i < 16; i++) {
      Ellipse e1 = new Ellipse(150, 100, 100, 50);
      e1.setStroke(Color.color(Math.random(), Math.random(),
Math.random()));
      e1.setFill(Color.WHITE); //不填充的效果更好，看右边
      e1.setRotate(i * 180 / 16);
      pane.getChildren().add(e1);
    }

    Scene scene = new Scene(pane, 300, 200);
    primaryStage.setTitle("ShowEllipse"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
}
```





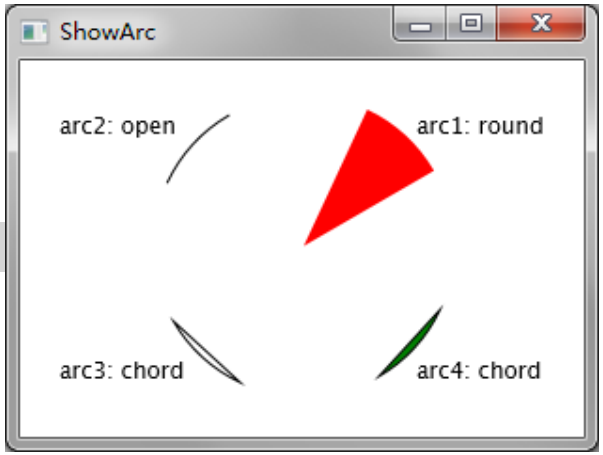
弧Arc

- 弧被定义为椭圆的局部，所以有些参数和椭圆是相同的。它的参数有这几个：centerX, centerY, radiusX, radiusY, startAngle, length, type (ArcType.OPEN, ArcType.CHORD, ArcType.ROUND)



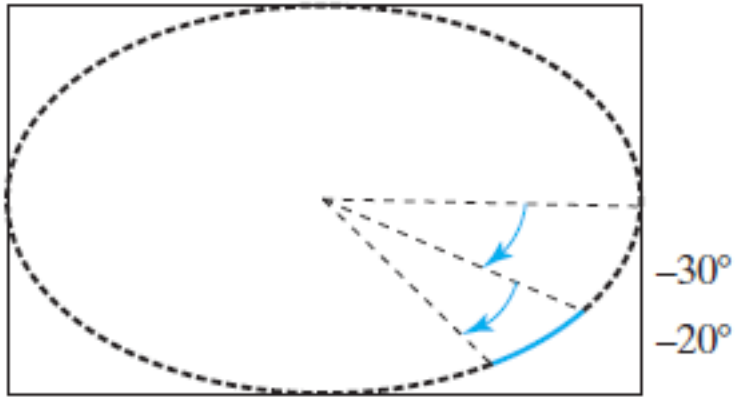
(a) `Arc(centerX, centerY, radiusX, radiusY, startAngle, length)`

```
Arc arc1 = new Arc(150, 100, 80, 80, 30, 35); // Create an arc
arc1.setFill(Color.RED); // Set fill color
arc1.setType(ArcType.ROUND); // Set arc type

Arc arc2 = new Arc(150, 100, 80, 80, 30 + 90, 35);
arc2.setFill(Color.WHITE);
arc2.setType(ArcType.OPEN);
arc2.setStroke(Color.BLACK);

Arc arc3 = new Arc(150, 100, 80, 80, 30 + 180, 35);
arc3.setFill(Color.WHITE);
arc3.setType(ArcType.CHORD);
arc3.setStroke(Color.BLACK);

Arc arc4 = new Arc(150, 100, 80, 80, 30 + 270, 35);
arc4.setFill(Color.GREEN);
arc4.setType(ArcType.CHORD);
arc4.setStroke(Color.BLACK);

// Create a group and add nodes to the group
Group group = new Group();
group.getChildren().addAll(new Text(210, 40, "arc1: round"),
  arc1, new Text(20, 40, "arc2: open"), arc2,
  new Text(20, 170, "arc3: chord"), arc3,
  new Text(210, 170, "arc4: chord"), arc4);
```

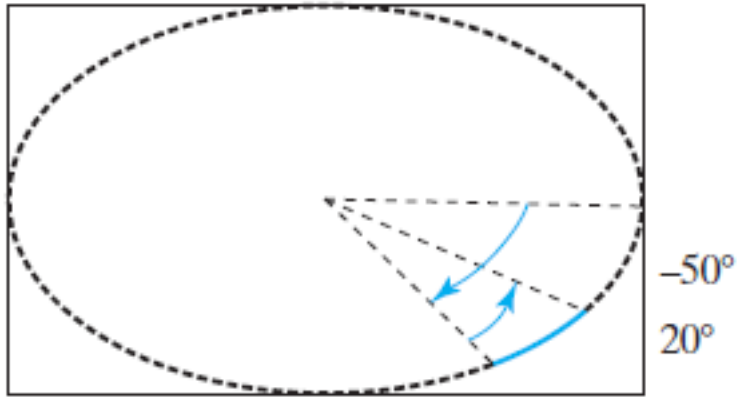


弧的角度也可以取负值

- 角度为负值表示顺时针，例如：
  - **new** Arc(x, y, radiusX, radiusY, **-30**, **-20**);
  - **new** Arc(x, y, radiusX, radiusY, **-50**, **20**);



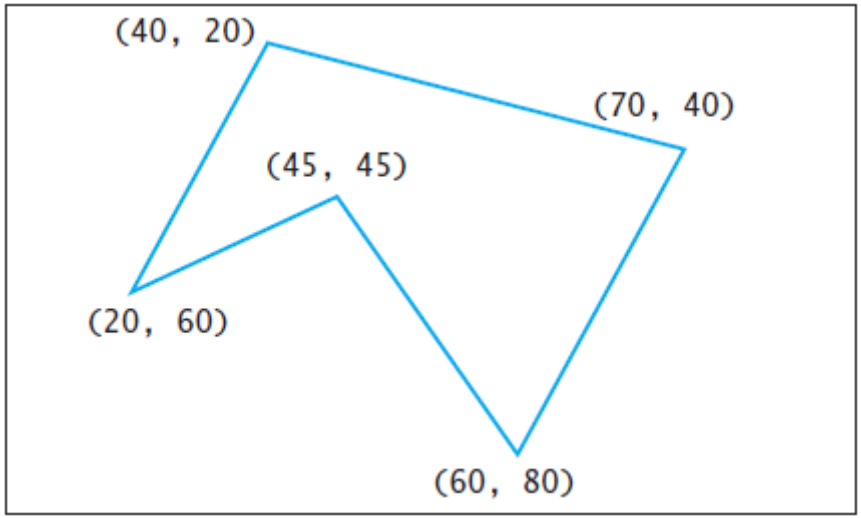
(a) Negative starting angle –30° and negative spanning angle –20°

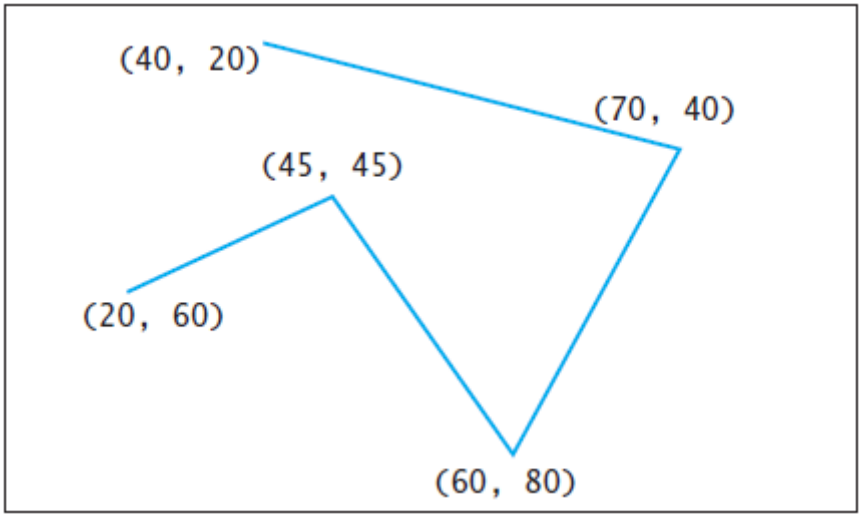


(b) Negative starting angle –50° and positive spanning angle 20°

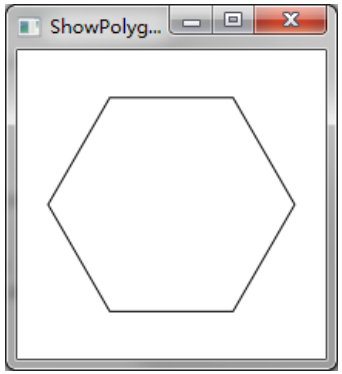
多边形和折线**Polygon** and **Polyline**

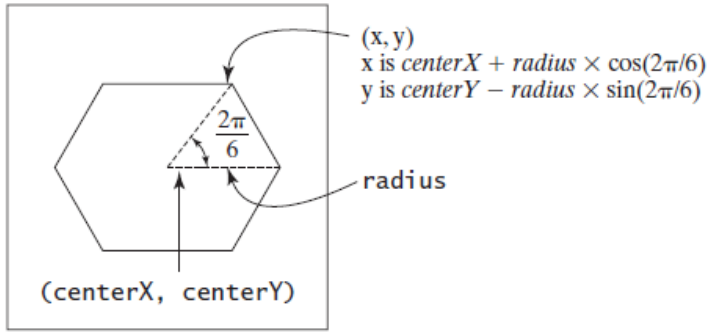
- 二者都是由点集构成，区别在于是否将起点和终点连线。所以它们都有一个成员方法getPoints()，用来返回一个可查看列表ObservableList<Double>，通过这个列表可以操作点集。





```
import javafx.application.Application;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.scene.shape.Polygon;
public class ShowPolygon extends Application {
  public void start(Stage primaryStage) {
  Pane pane = new Pane();
  Polygon polygon = new Polygon();
  pane.getChildren().add(polygon);
  polygon.setFill(Color.WHITE);
  polygon.setStroke(Color.BLACK);
  ObservableList<Double> list = polygon.getPoints();
  final double WIDTH = 200, HEIGHT = 200;
  double centerX = WIDTH / 2, centerY = HEIGHT / 2;
  double radius = Math.min(WIDTH, HEIGHT) * 0.4;
  for (int i = 0; i < 6; i++) {
    list.add(centerX + radius * Math.cos(2 * i * Math.PI / 6));
    list.add(centerY - radius * Math.sin(2 * i * Math.PI / 6));
  }
  Scene scene = new Scene(pane, WIDTH, HEIGHT);
  primaryStage.setTitle("ShowPolygon"); primaryStage.setScene(scene);
  primaryStage.show();
} }
```





THE END