# Lecture5 GUI 和 JavaFX

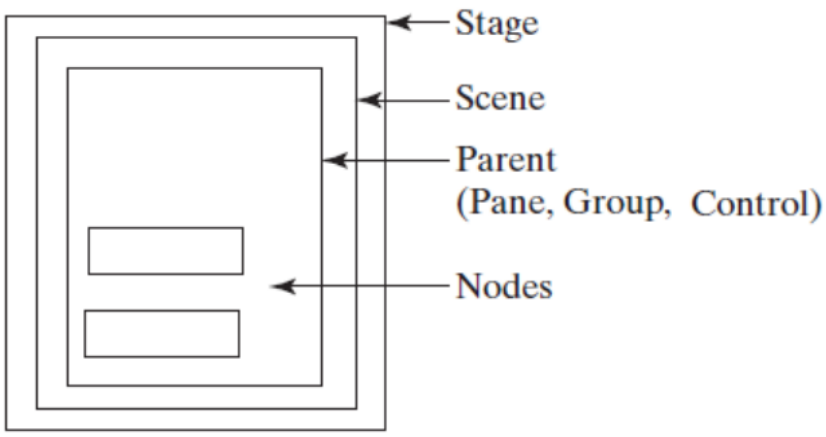
## 1. JavaFX 介绍

`javafx.application.Application` 定义了编写 JavaFX 程序的基本框架

### 第一个 JavaFX 程序

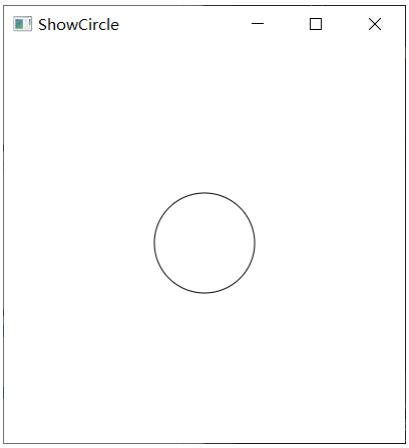
```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.layout.StackPane;
4  import javafx.stage.Stage;
5
6  public class MyJavaFX extends Application { // 继承 Application
7
8      @Override // 重写 Application 类的 start 方法
9      public void start(Stage primaryStage) throws Exception {
10         StackPane pane = new StackPane(); // 创建一个 Pane
11         pane.getChildren().add(new javafx.scene.control.Button("ok")); // 向 Pane
   添加一个 btn
12         Scene scene = new Scene(pane, 500, 200); // 将 Pane 加入 Scene 里
13         primaryStage.setTitle("MyJavaFX");
14         primaryStage.setScene(scene); // 将 Scene 加入 Stage 里
15         primaryStage.show(); // 展示 Stage
16     }
17
18     // main 方法执行 Application
19     public static void main(String[] args) {
20         Application.launch();
21     }
22 }
```

## 架构




- `Stage` 是一个窗口，它可以展示一些包含了 `Node` 的 `Scene`
- `Node` 可以是 `Shape`，`ImageView`，`Control`，`Group` 和 `Pane`
- 一个 JavaFX 程序可以展示多个 `Stage`

## 画一个圆形



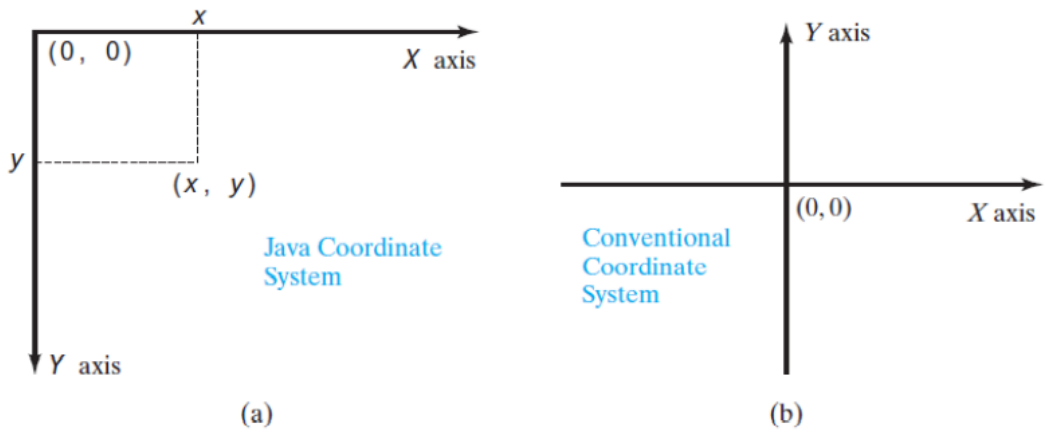

```java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;

public class ShowCircle extends Application {


    @Override
    public void start(Stage primaryStage) throws Exception {
        // Node
        Circle circle = new Circle();
        circle.setCenterX(200);
        circle.setCenterY(200);
        circle.setRadius(50);
        circle.setStroke(Color.BLACK);
```

```
19          circle.setFill(Color.WHITE);
20
21          // Pane
22          Pane pane = new Pane();
23          pane.getChildren().add(circle);
24
25          // Scene
26          Scene scene = new Scene(pane, 400, 400);
27
28          // Stage
29          primaryStage.setTitle("ShowCircle");
30          primaryStage.setScene(scene);
31          primaryStage.show();
32      }
33
34      public static void main(String[] args) {
35          Application.launch();
36      }
37  }
```

- 在 Java 中，坐标系系统的度量单位是像素，(0, 0) 是在左上角

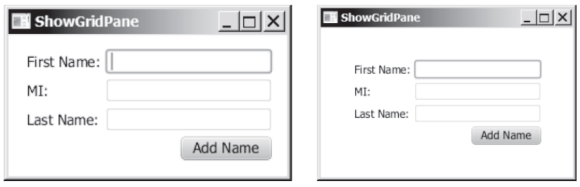
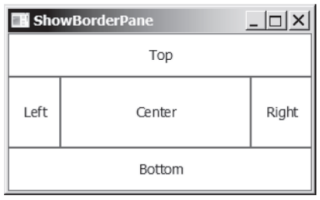
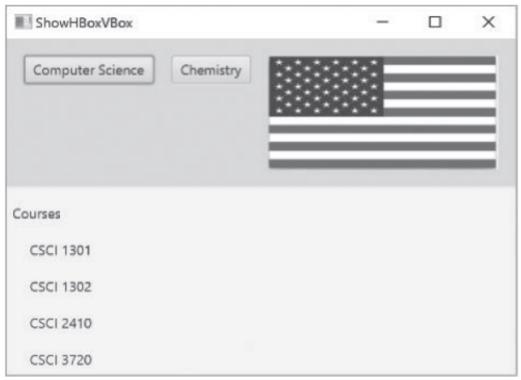



## 2. Pane 和 Group 的布局

JavaFX提供了多种类型的 `Pane`，用于在所需的位置和大小中自动布局 `Node`

`Pane` 和 `Group` 是保存节点的容器

- `Group` 类通常用于将节点分组，并作为一个组执行转换和扩展
- `Pane` 和 `UI Control` 对象是可调整大小的
- `Group`、`Shape` 和 `Text` 对象是不可调整大小的
- JavaFX 提供了许多类型的 `Pane` 来组织容器中的 `Node`

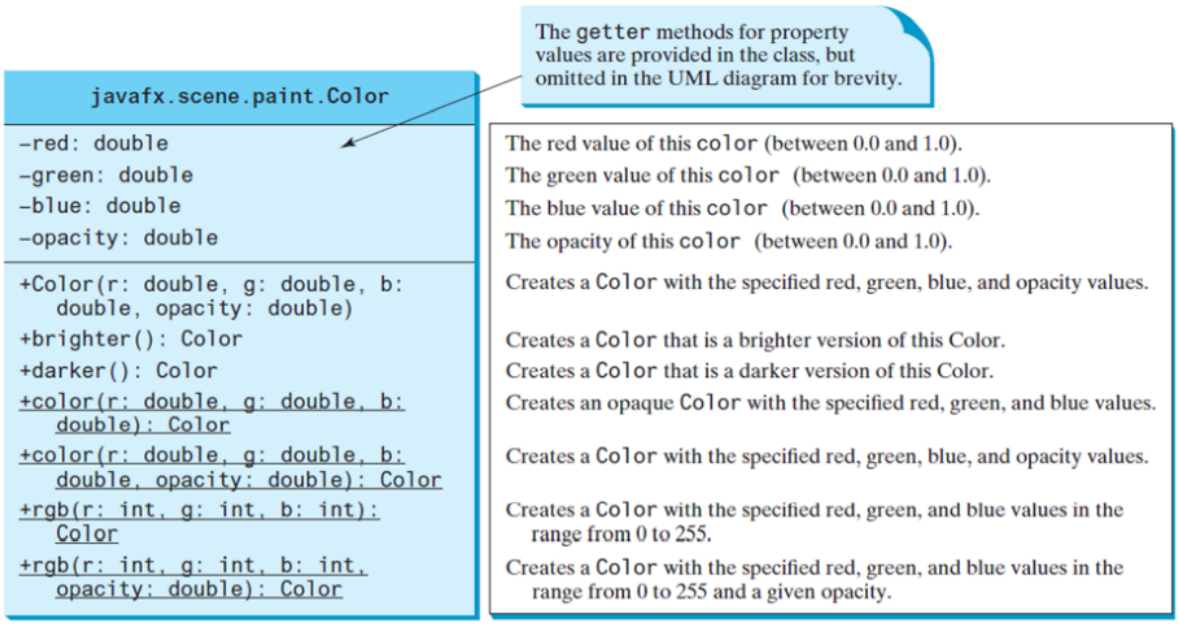
| 类型 | 描述 | 图片描述 |
| --- | --- | --- |
| Pane | `Pane` 的基类，它包含 `getChildren()` 方法，用于返回窗格中的节点列表 |  <br>The nodes fill in the rows in the `FlowPane` one after another. |
| StackPane | 将 `Node` 一个一个地放在窗格的中心 | |
| FlowPane | 将 `Node` 按行水平放置或按列垂直放置 | |
| GridPane | 将 `Node` 放置在二维网格中的单元中 |  <br>The `GridPane` places the nodes in a grid with a specified column and row indices. |
| BorderPane | 将 `Node` 放置在顶部、右侧、底部、左侧和中心区域 |  <br>The `BorderPane` places the nodes in five regions of the pane. |
| HBox | 将 `Node` 放在单行中 |  <br>The `HBox` places the nodes in one row, and the `VBox` places the nodes in one column. |
| VBox | 将 `Node` 放在单列中 | |

# 3. JavaFX 相关类

## Color 类

`Color` 类用来创建颜色

JavaFX 定义了一个抽象的 `Paint` 类来描述绘制一个 `Node` 的情况，`javafx.scene.paint.Color` 是 `Paint` 的一个具体实现子类




除了 RGB 颜色系统之外，`Color` 类也支持一种 HSB 系统，在 HSB 系统中，一个颜色由 3 种属性定义

| | 色彩 Hue | 饱和度 Saturation | 亮度 Brightness |
|---|---|---|---|
| 数据类型 | double | double | double |
| 取值范围 | 0.0 - 360.0 | 0.0 - 1.0 | 0.0 - 1.0 |

- 色彩的值是以度来表示的，颜色被看作是沿着一个圆排列的

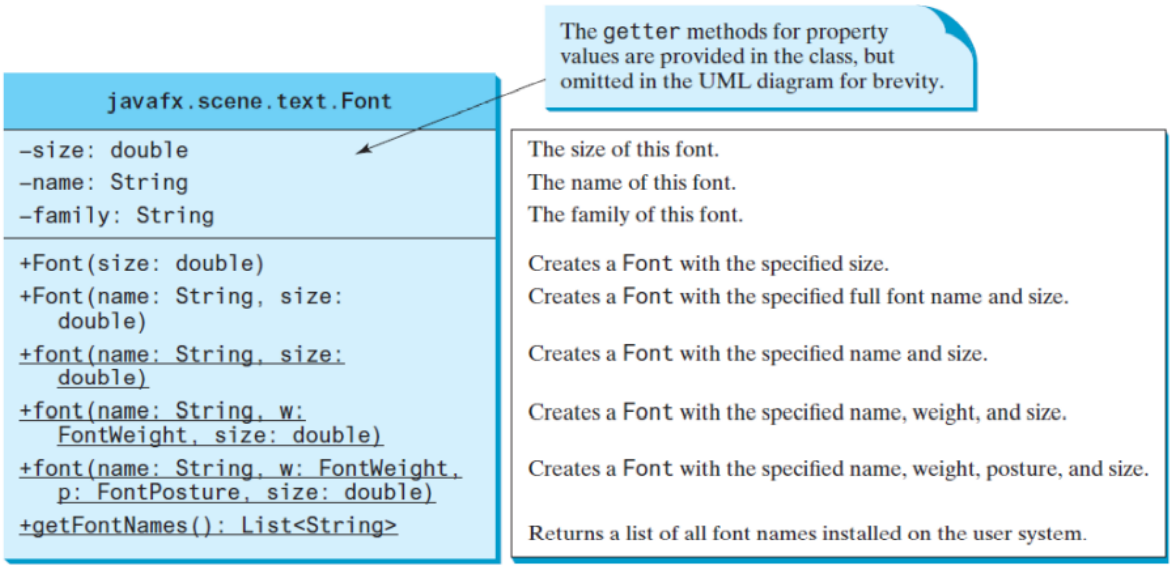
`Color` 类提供静态方法 `Color.hsb(h,s,b)` 和 `Color.hsb(h,s,b,a)` 来生成 HSB 颜色

```
1   Color randomColor = Color.hsb(360*Math.random(), 1.0, 1.0)
```

RGB系统和HSB系统只是描述同一组颜色的不同方式，可以在一个系统和另一个系统之间进行转换

## Font 类

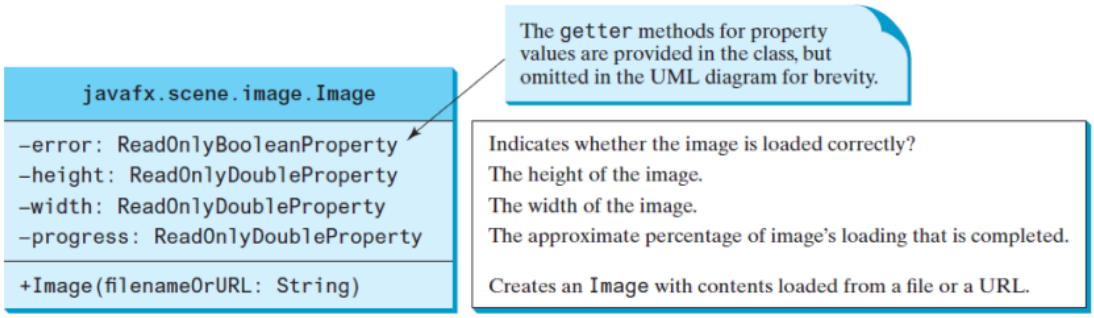
`Font` 类描述字体名称、字号和大小

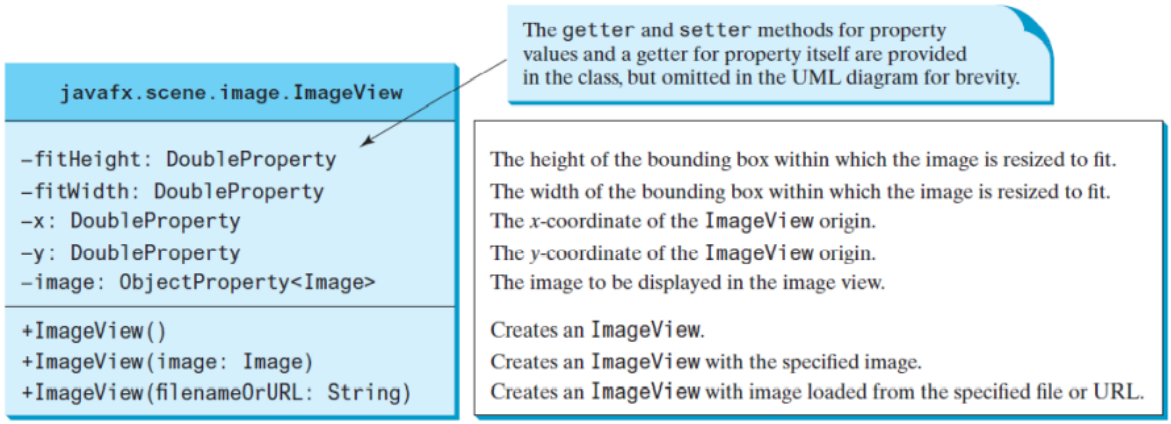



## Image 和 ImageView 类

`Image` 类表示一个图形图像， `ImageView` 类可以用来显示一个图像

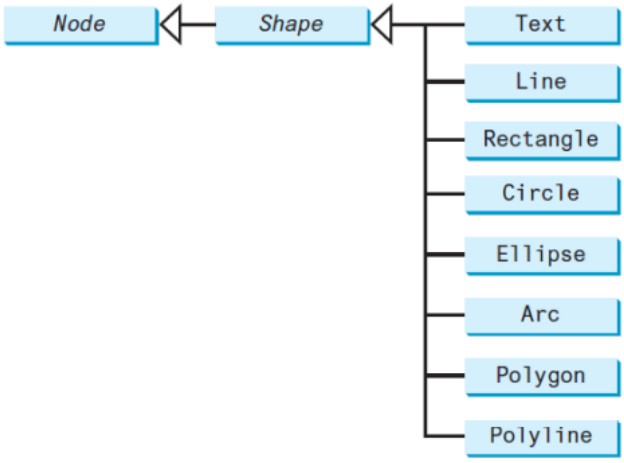
```
1   Image image = new Image("image.gif");
2   ImageView img = new ImageView(image);
3   ImageView img2 = new ImageView("image.gif");
```



## Shape 类

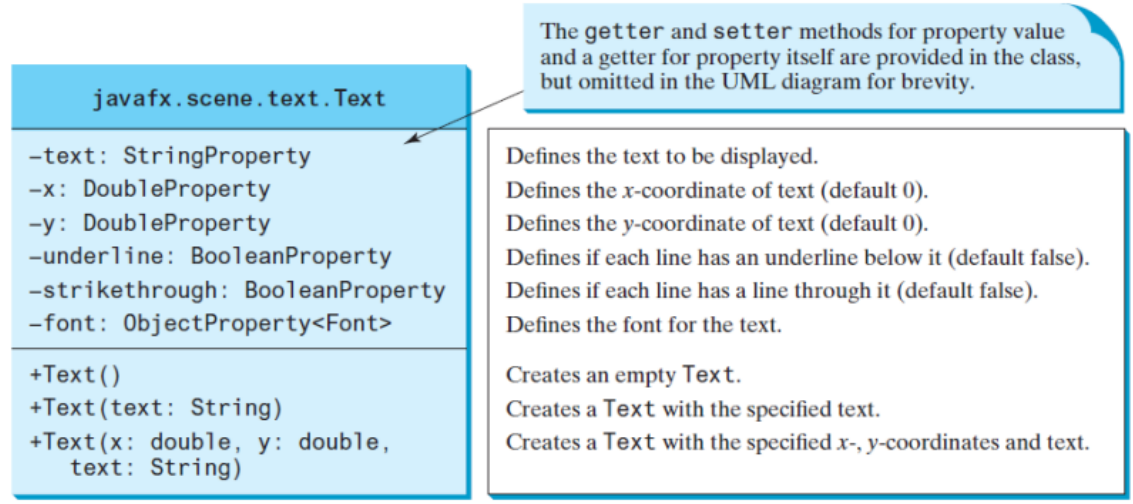
JavaFX 提供了许多 `Shape` 类，用于绘制文本、线条、圆、矩形、椭圆、圆弧、多边形和折线

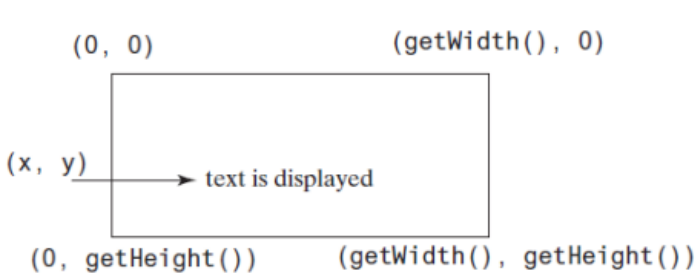



`Shape` 类是一个抽象基类，它表示了所有形状的公有特性，其中典型的有

- `fill()`：指定一个颜色填充形状的内部
- `stroke()`：指定一个颜色填充形状的轮廓
- `strokeWidth()`：指定形状边缘的宽度

## Text 类

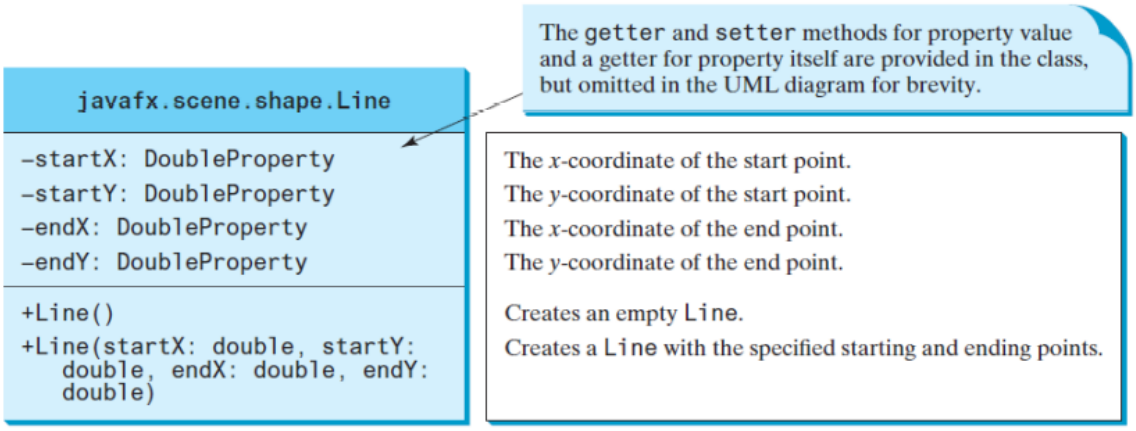


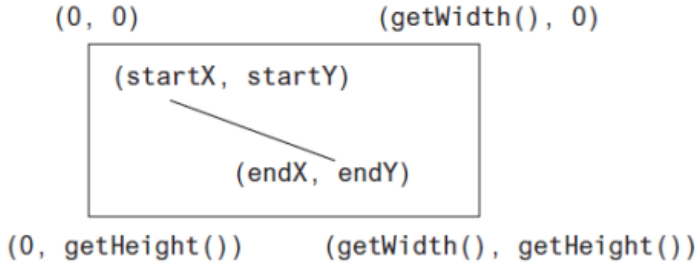




(a) Text(x, y, text)

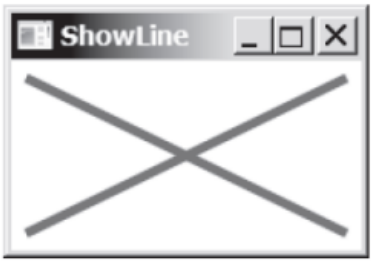

(b) Three Text objects are displayed

## Line 类



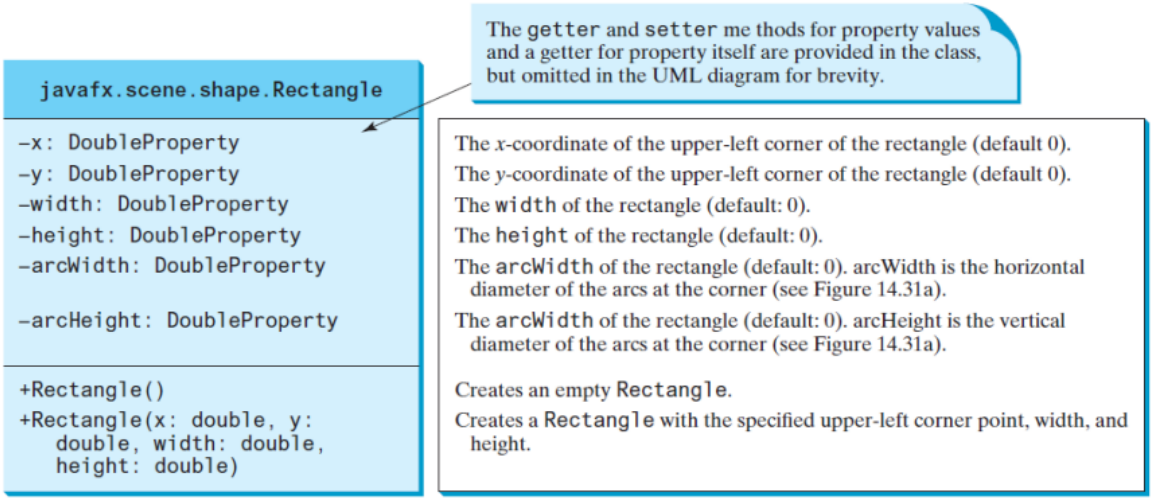



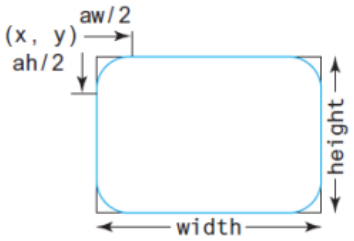

(a) Line(startX, startY, endX, endY)



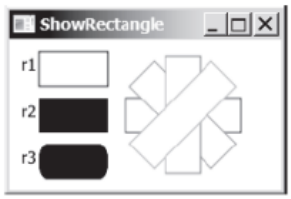

(b) Two lines are displayed across the pane.

## Rectangle 类

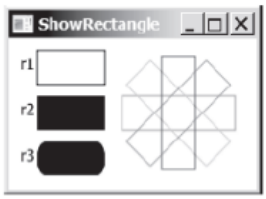



(a) Rectangle(x, y, w, h)

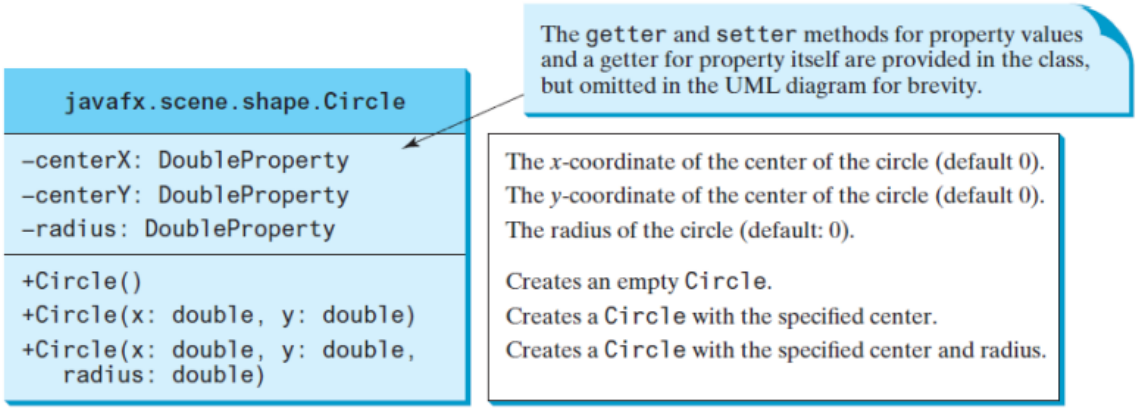



(b) Multiple rectangles are displayed.

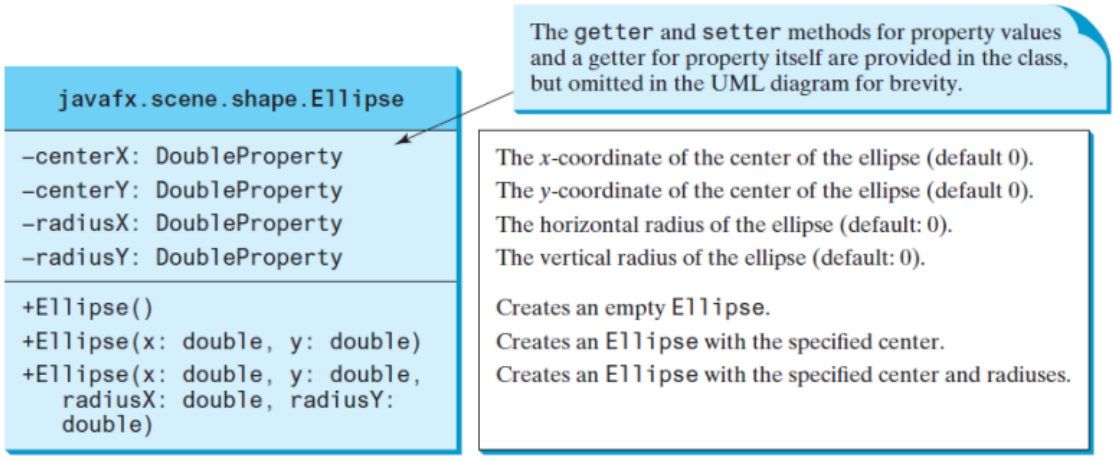


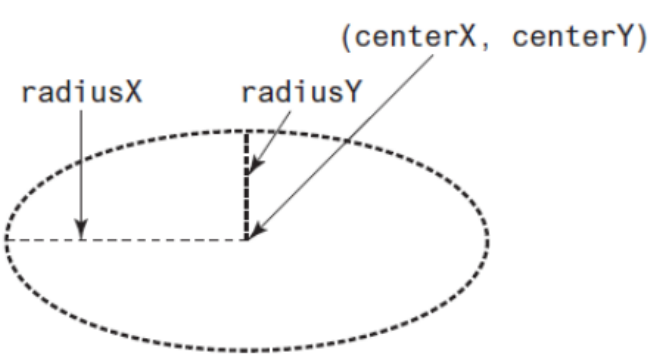

(c) Transparent rectangles are displayed.

## Circle 类

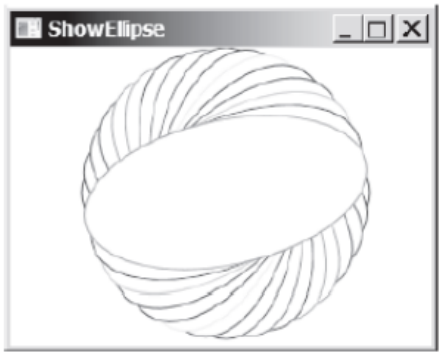



**Ellipse 类**




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




(b) Multiple ellipses are displayed.

**Arc 类**

The `getter` and `setter` methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

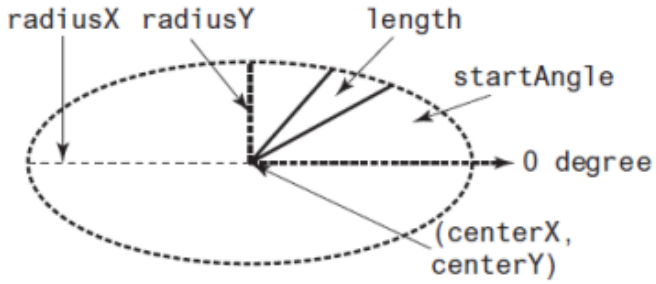
**javafx.scene.shape.Arc**

```
-centerX: DoubleProperty
-centerY: DoubleProperty
-radiusX: DoubleProperty
-radiusY: DoubleProperty
-startAngle: DoubleProperty
-length: DoubleProperty
-type: ObjectProperty<ArcType>
```

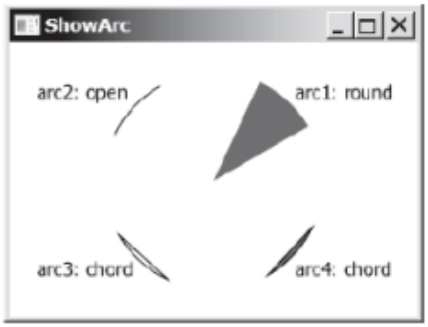
```
+Arc()
+Arc(x: double, y: double,
   radiusX: double, radiusY:
   double, startAngle: double,
   length: double)
```

The *x*-coordinate of the center of the ellipse (default 0).
The *y*-coordinate of the center of the ellipse (default 0).
The horizontal radius of the ellipse (default: 0).
The vertical radius of the ellipse (default: 0).
The start angle of the arc in degrees.
The angular extent of the arc in degrees.
The closure type of the arc (`ArcType.OPEN`, `ArcType.CHORD`, `ArcType.ROUND`).

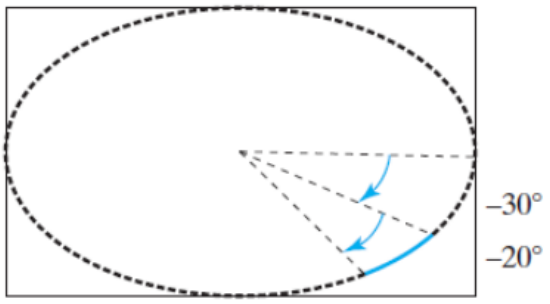
Creates an empty `Arc`.
Creates an `Arc` with the specified arguments.



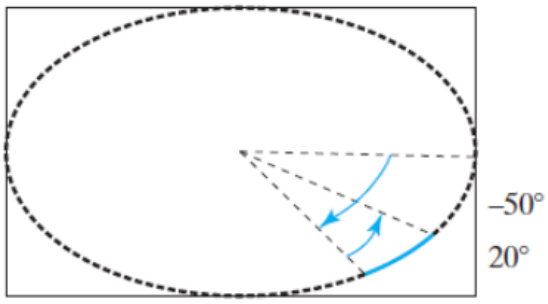

(a) `Arc(centerX, centerY, radiusX, radiusY, startAngle, length)`




(b) Multiple ellipses are displayed.




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




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

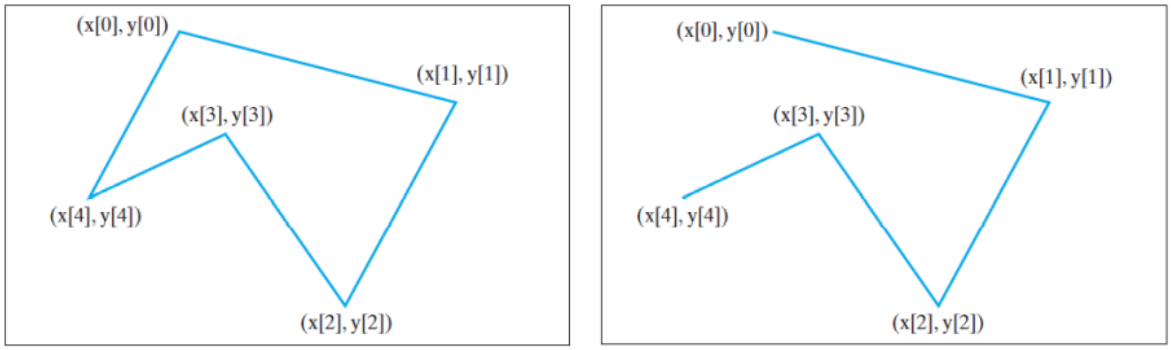
Angles may be negative.

### Polygon 类和 Polyline 类

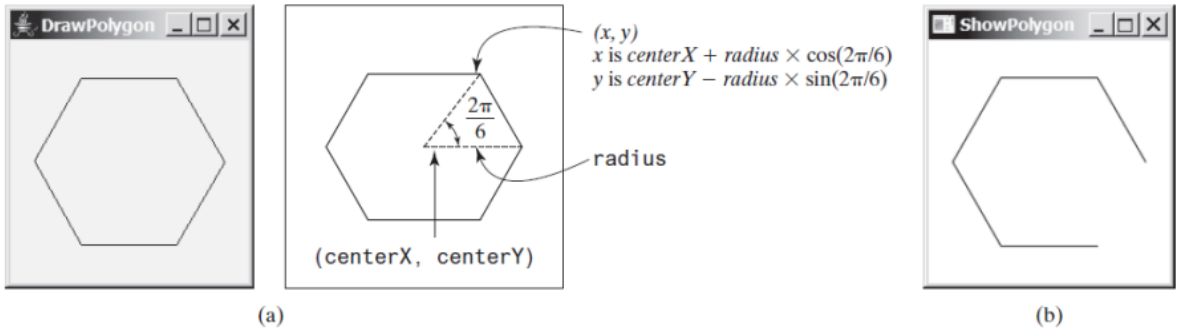
**javafx.scene.shape.Polygon**

```
+Polygon()
+Polygon(double... points)
+getPoints():
   ObservableList<Double>
```

Creates an empty `Polygon`.
Creates a `Polygon` with the given points.
Returns a list of double values as *x*- and *y*-coordinates of the points.




`Polygon` is closed and `Polyline` is not closed.




(a) A `Polygon` is displayed. (b) A `Polyline` is displayed.

# 4. 属性绑定 Property Binding

可以将一个 `target` 对象绑定在一个 `source` 对象上

对 `source` 对象发生的改变将会自动影响到 `target` 对象

- `target` 对象也叫做：binding object 或者 binding property
- `source` 对象也叫做：bindable object 或者 observable object

在第一部分讲到的画一个圆形的代码，现在可以更改部分代码

```
    @Override
    public void start(Stage primaryStage) throws Exception {
        // Pane
        Pane pane = new Pane();
        // Node
        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);

```

```
15          // Scene
16          Scene scene = new Scene(pane, 400, 400);
17
18          // Stage
19          primaryStage.setTitle("ShowCircle");
20          primaryStage.setScene(scene);
21          primaryStage.show();
22      }
```

下面两条语句完全等价

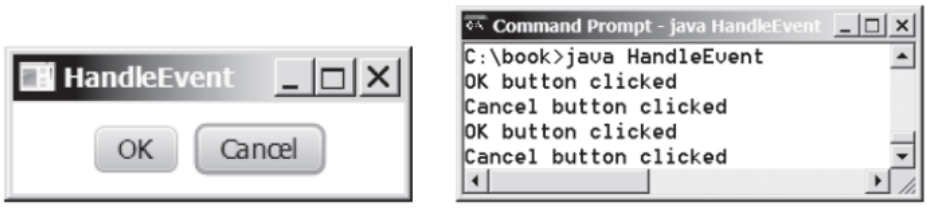
```
1   circle.centerXProperty().bind(pane.widthProperty().divide(2));
```

```
1   DoubleProperty centerX = circle.centerXproperty();
2   DoubleProperty width = pane.widthProperty();
3   centerX.bind(width.divide(2));
```

由于 `centerX` 与 `width.divide(2)` 绑定，当 `pane` 的大小改变的时候，`centerX` 会自动将自己更新，来匹配 `pane` 的宽度

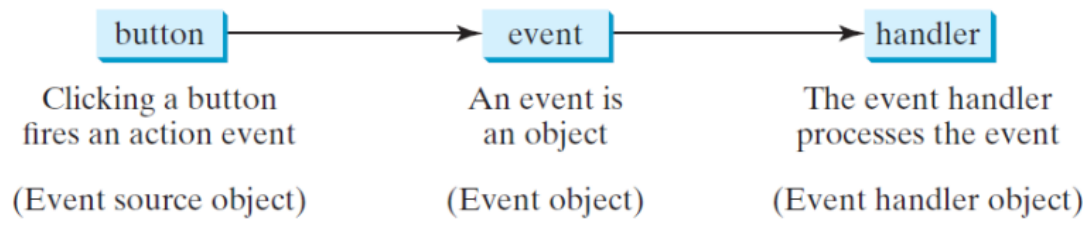
# 5. 事件驱动的编程

## 基本介绍




(a) (b)

- 程序展示了两个 btn
- 当点击 btn 的时候，命令行会显示一些消息




当点击 btn 的时候，发送一个事件，这个事件被事件处理器处理

## 处理事件

并不是所有的对象都可以处理事件，为了能进行处理，必须满足下面两个条件

- 对象必须是 `EventHandler<T extends Event>` 接口的实例化， `T` 是 `Event` 的子类
- `EventHandler` 对象的 `handler` 必须与一个产生事件源的对象进行绑定，使用方法 `source.setOnAction(hanlder)`

注意，由于 `EventHandler` 接口是一个函数式接口，它里面只有一个抽象方法 `handler()` ，所以可以写成 lambda 表达式

### 示例

```
1  import javafx.application.Application;
2  import javafx.geometry.Pos;
3  import javafx.scene.Scene;
4  import javafx.scene.control.Button;
5  import javafx.scene.layout.HBox;
6  import javafx.stage.Stage;
7
8
9  public class HandleEvent extends Application {
10     @Override
11     public void start(Stage primaryStage) throws Exception {
12         HBox pane = new HBox(10);
13         pane.setAlignment(Pos.CENTER);
14         Button btOk = new Button("OK");
15         Button btCancel = new Button("Cancel");
16         btOk.setOnAction(event -> System.out.println("Ok btn clicked"));
17         btCancel.setOnAction(event -> System.out.println("cancel btn clicked"));
18         pane.getChildren().addAll(btOk, btCancel);
19         Scene scene = new Scene(pane);
20         primaryStage.setTitle("Handle Event");
21         primaryStage.setScene(scene);
22         primaryStage.show();
23     }
24
25     public static void main(String[] args) {
26        Application.launch();
27     }
28 }
```

## Event 类

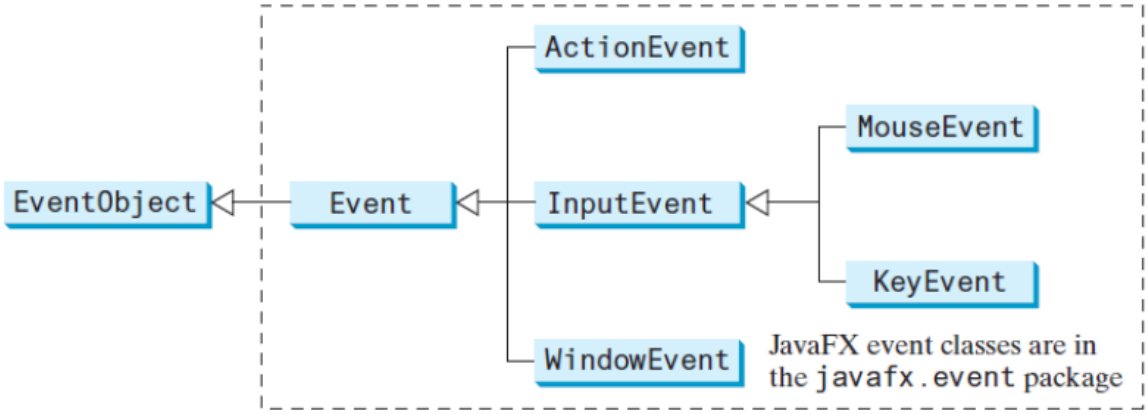



**TABLE 15.1** User Action, Source Object, Event Type, Handler Interface, and Handler

| *User Action* | *Source Object* | *Event Type Fired* | *Event Registration Method* |
|---|---|---|---|
| Click a button | `Button` | `ActionEvent` | `setOnAction(EventHandler<ActionEvent>)` |
| Press Enter in a text field | `TextField` | `ActionEvent` | `setOnAction(EventHandler<ActionEvent>)` |
| Check or uncheck | `RadioButton` | `ActionEvent` | `setOnAction(EventHandler<ActionEvent>)` |
| Check or uncheck | `CheckBox` | `ActionEvent` | `setOnAction(EventHandler<ActionEvent>)` |
| Select a new item | `ComboBox` | `ActionEvent` | `setOnAction(EventHandler<ActionEvent>)` |
| Mouse pressed | `Node, Scene` | `MouseEvent` | `setOnMousePressed(EventHandler<MouseEvent>)` |
| Mouse released | | | `setOnMouseReleased(EventHandler<MouseEvent>)` |
| Mouse clicked | | | `setOnMouseClicked(EventHandler<MouseEvent>)` |
| Mouse entered | | | `setOnMouseEntered(EventHandler<MouseEvent>)` |
| Mouse exited | | | `setOnMouseExited(EventHandler<MouseEvent>)` |
| Mouse moved | | | `setOnMouseMoved(EventHandler<MouseEvent>)` |
| Mouse dragged | | | `setOnMouseDragged(EventHandler<MouseEvent>)` |
| Key pressed | `Node, Scene` | `KeyEvent` | `setOnKeyPressed(EventHandler<KeyEvent>)` |
| Key released | | | `setOnKeyReleased(EventHandler<KeyEvent>)` |
| Key typed | | | `setOnKeyTyped(EventHandler<KeyEvent>)` |