Week3
Event-Driven Programming

Dr. Abdul Razzaq

Abdul.Razzaq@ul.ie

Agenda

- **Event-driven Model**
- **Model View Controller (MVC) Architecture**
- **JavaFX – Designing GUI in Java**

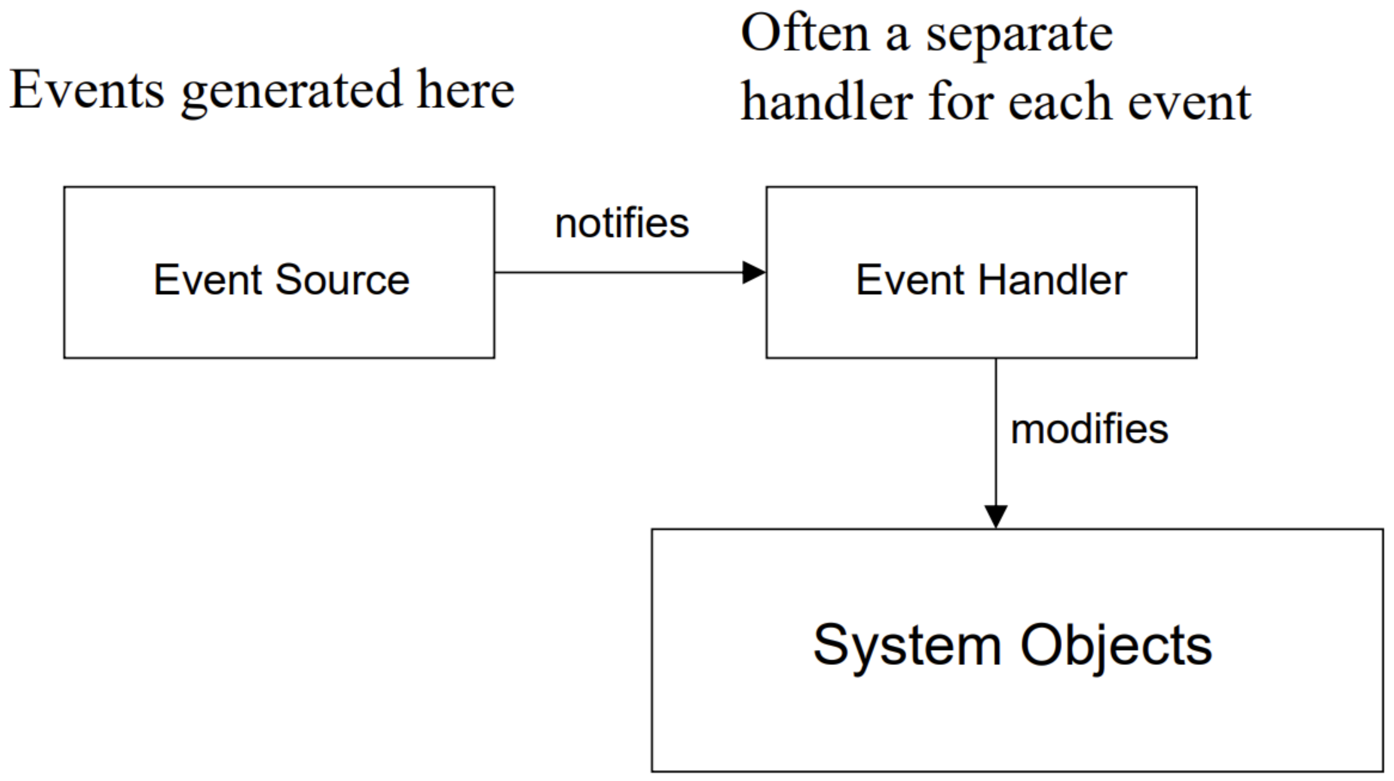
Event

- An event can be defined as a ***type of signal*** to the program that ***something has happened***.
- The event is generated by external user actions such as:
  - *Mouse Click*
  - *Key Press*
  - *System Clock Timer*

Event Handler

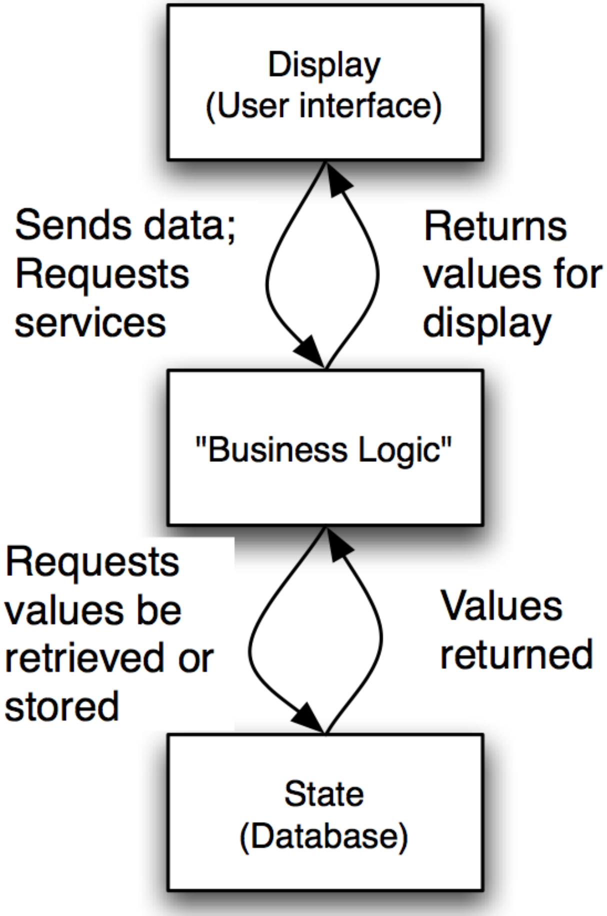
- Events TRIGGER response
- Event Handler implements a *response* to an event
- *Action or Sequence of Actions carried out when Event Occurs (Methods)*
- Programmers do not have to think of all possible events but must plan a response for each event they are interested in (Event Handler)
- Linking the Event with its Handler
  - Event Handling Methods must be ***linked*** to the Specific Event
  - Event Occurs which ***causes*** the Event Handler Method to execute

Basic Event-driven Model



State-Logic-Display: Three-Tiered Pattern

- **Application Examples**
  - □ **Business applications**
  - □ **Multi-player games**
  - □ **Web-based applications**


Display
(User interface)
Sends data; Requests services
Returns values for display
"Business Logic"
Requests values be retrieved or stored
Values returned
State
(Database)

Software Architecture: Foundations, Theory, and Practice; Richard N. Taylor, Nenad Medvidovic, and Eric M. Dashofy; © 2008 John Wiley & Sons, Inc. Reprinted with permission.

MVC

- **Model: data, user model**
  - **Largely: domain classes + containers**
- **View: display the data**
  - **May also include handling user actions**
  - **Often: app contains multiple views**
- **Controller: business logic, interface to model**
  - **May include handling user actions**

- **Advantages:**
  - **Distributed logic**
    - Separate messy display code from the domain implementation
  - **Supports automated testing of model, business logic**
- **Challenges:**
  - **Not clear what functionality goes where**
  - **Confusing to novice programmers**
- **When to use?**
  - **When you want maintainable apps!**

Model-View-Controller (MVC)

- Objective: ***Separation between information, (user) interaction, and presentation***.
- When a model object value changes, a notification is sent to the view and the controller. So that the view can update itself and the controller can modify the view of its logic as required.
- When handling input from the user the windowing system sends the user event to the controller; If a change is required, the controller updates the model object.

UNIVERSITY
of
LIMERICK
OLLSCOIL LUIMNIGH

Designing GUI in Java

Dr Abdul Razzaq

abdul.razzaq@ul.ie

www.ul.ie

Some content is adopted from Jan Bodnar

UNIVERSITY *of* LIMERICK
OLLSCOIL LUIMNIGH

JavaFX Part

1 **Introduction to JavaFX**

1.1 Hello World of JavaFX
1.2 Structure of JavaFX

2 **JavaFX layout**

FlowPane, HBox, BorderPane, AnchorPane, GridPane

3 **Designing a UI**

3.1 JavaFX Shapes
3.2 JavaFX Controls

4 **Event Driven Programming**

3.1 Procedural vs Event-Driven Programming
3.2 How to make Buttons click?

Kinds of User Interfaces

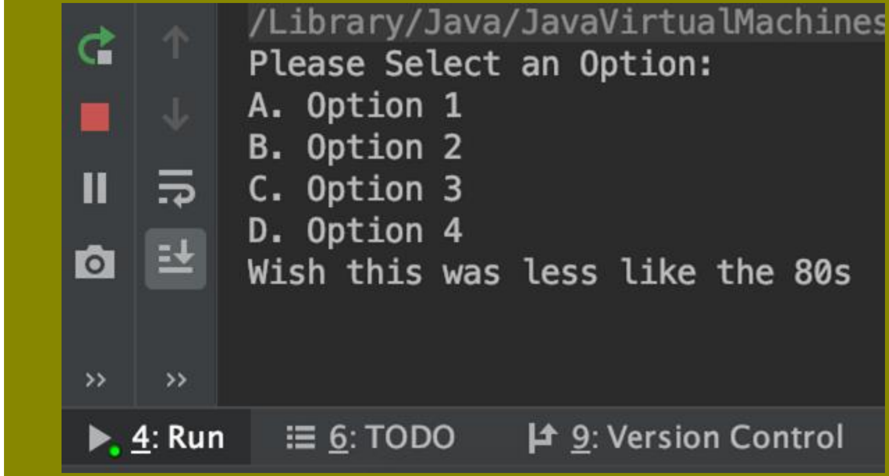
Textual

- Uses text and keyboard
- Driven by:
  - text input prompts
  - command-line interfaces
- Relatively easy to program

Graphical

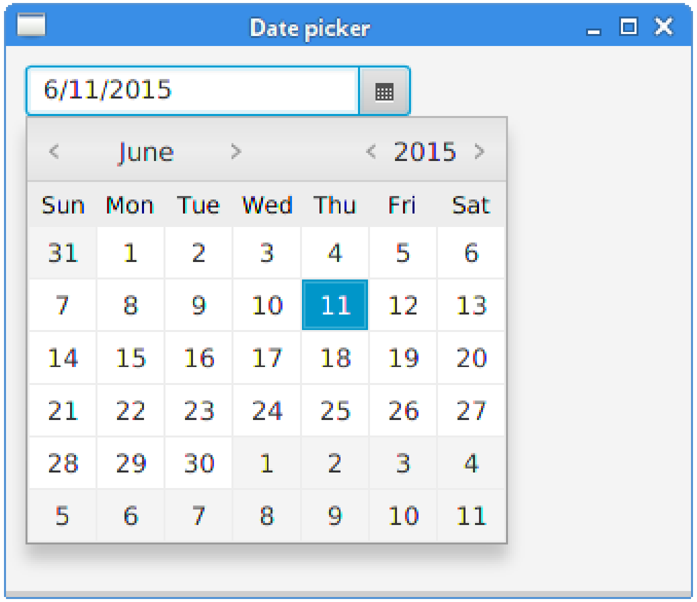
- Uses pictures and mouse (in addition to text & keyboard)
- Driven by user-initiated graphical events, e.g.,
  - pressing mouse button
  - releasing mouse button
  - dragging the mouse
  - pressing a keyboard key
- Requires more programming (mostly using libraries)

What you can do now -



```
/Library/Java/JavaVirtualMachines
Please Select an Option:
A. Option 1
B. Option 2
C. Option 3
D. Option 4
Wish this was less like the 80s
```

What you will be able to do after learning JavaFX-



6/11/2015

< June > < 2015 >

Sun	Mon	Tue	Wed	Thu	Fri	Sat
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	1	2	3	4
5	6	7	8	9	10	11

How do we design GUI?

Use GUI libraries:

AWT-> When Java was introduced, the GUI classes were bundled in a library known as the Abstract Windows Toolkit (AWT).

Swing-> Swing components are painted directly on canvases using Java code. Swing components depend less on the target platform and use less of the native GUI resource.

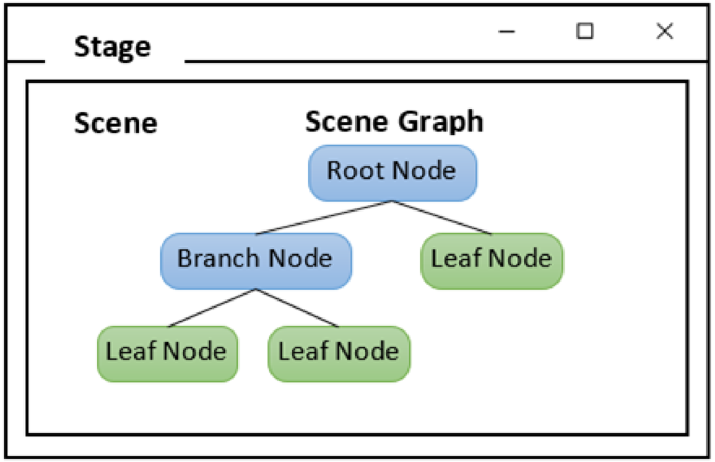
JavaFX-> Swing and AWT are replaced by the JavaFX platform for developing rich GUI/Internet applications from Java 8 onwards.

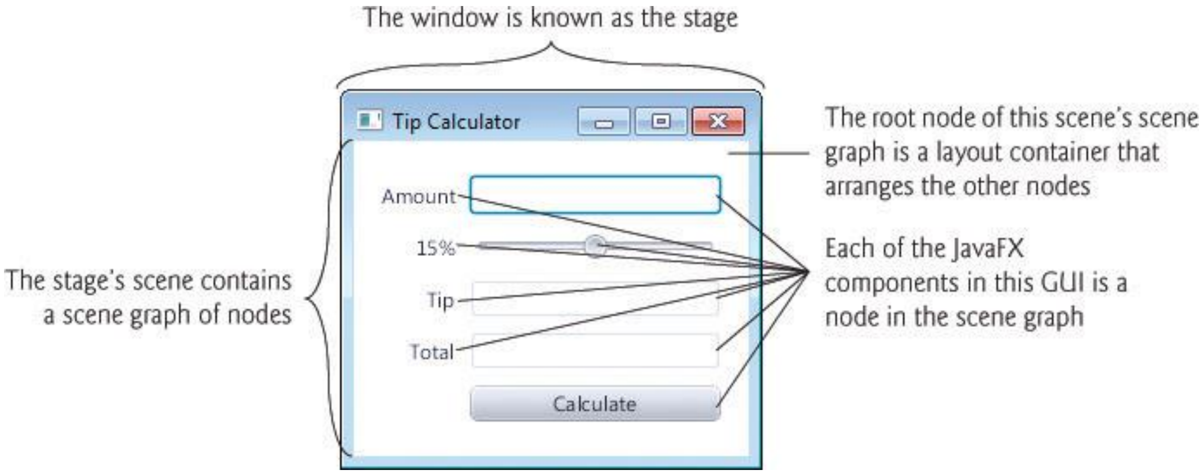
Setting up JavaFX - openjfx.io

To set up JavaFX on Windows with Netbeans, follow the instructions on the following link: https://openjfx.io/openjfx-docs/

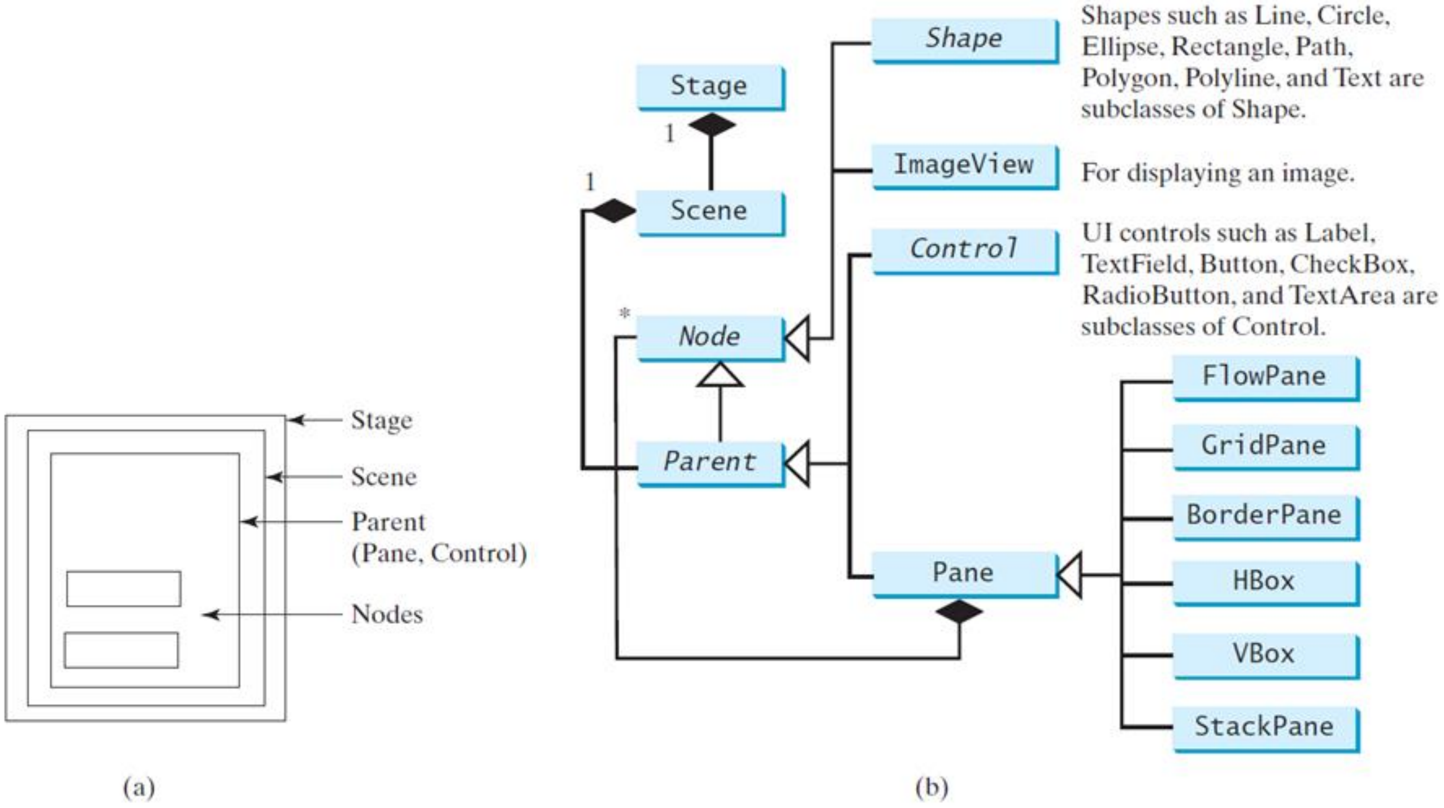
Non-modular with Maven is Preferred: Easy

Basic Structure of JavaFX



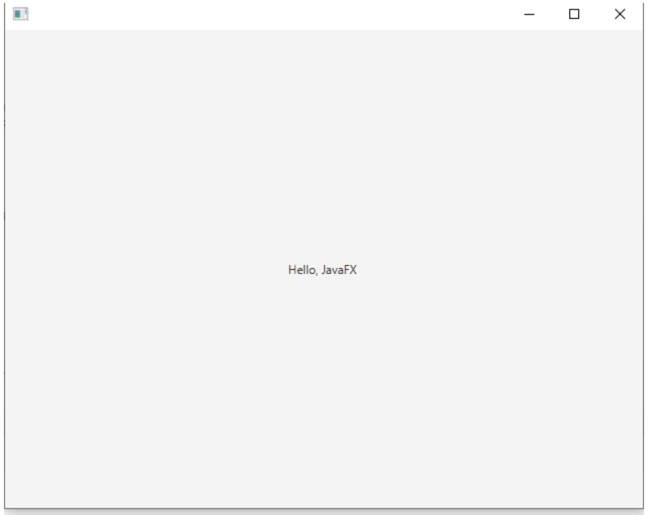

The window is known as the stage
Tip Calculator
Amount
15%
Tip
Total
Calculate
The root node of this scene's scene graph is a layout container that arranges the other nodes
Each of the JavaFX components in this GUI is a node in the scene graph
The stage's scene contains a scene graph of nodes

Basic Structure of JavaFX


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

Basic Code Structure of JavaFX

- **Every JavaFX-based system must extend Application**
- **It must override the start(Stage) method**
  - **Many IDEs need the launch method explicitly invoked**



Hello, JavaFX

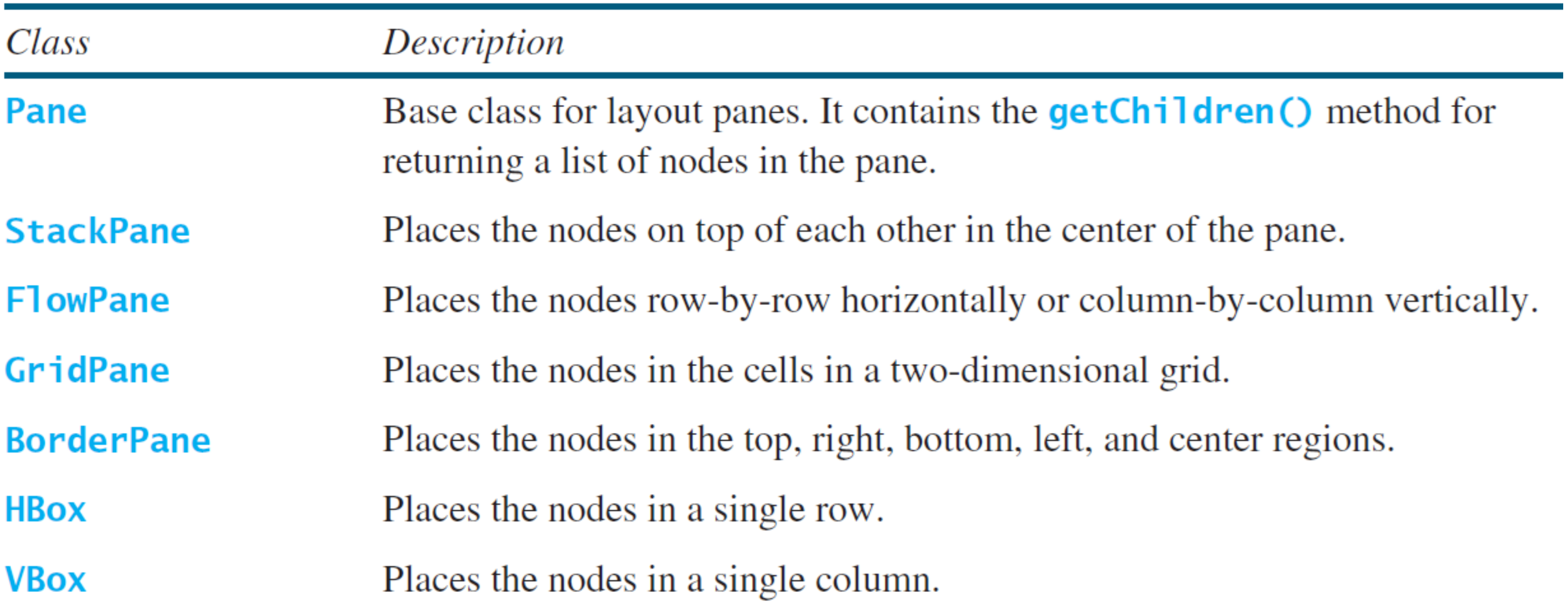
```
package com.mycompany.hellofx;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class App extends Application {
    @Override
    public void start(Stage stage) {
       var label = new Label("Hello, JavaFX ");
        var scene = new Scene(new StackPane(label),
                   640, 480);
        stage.setScene(scene);
        stage.show();
    }

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

UNIVERSITY of LIMERICK
OLLSCOIL LUIMNIGH

JavaFX layout panes

Layout panes are containers that are used for flexible and dynamic arrangements of UI controls within a scene graph of a JavaFX application. As a window is resized, the layout pane automatically repositions and resizes the nodes it contains.

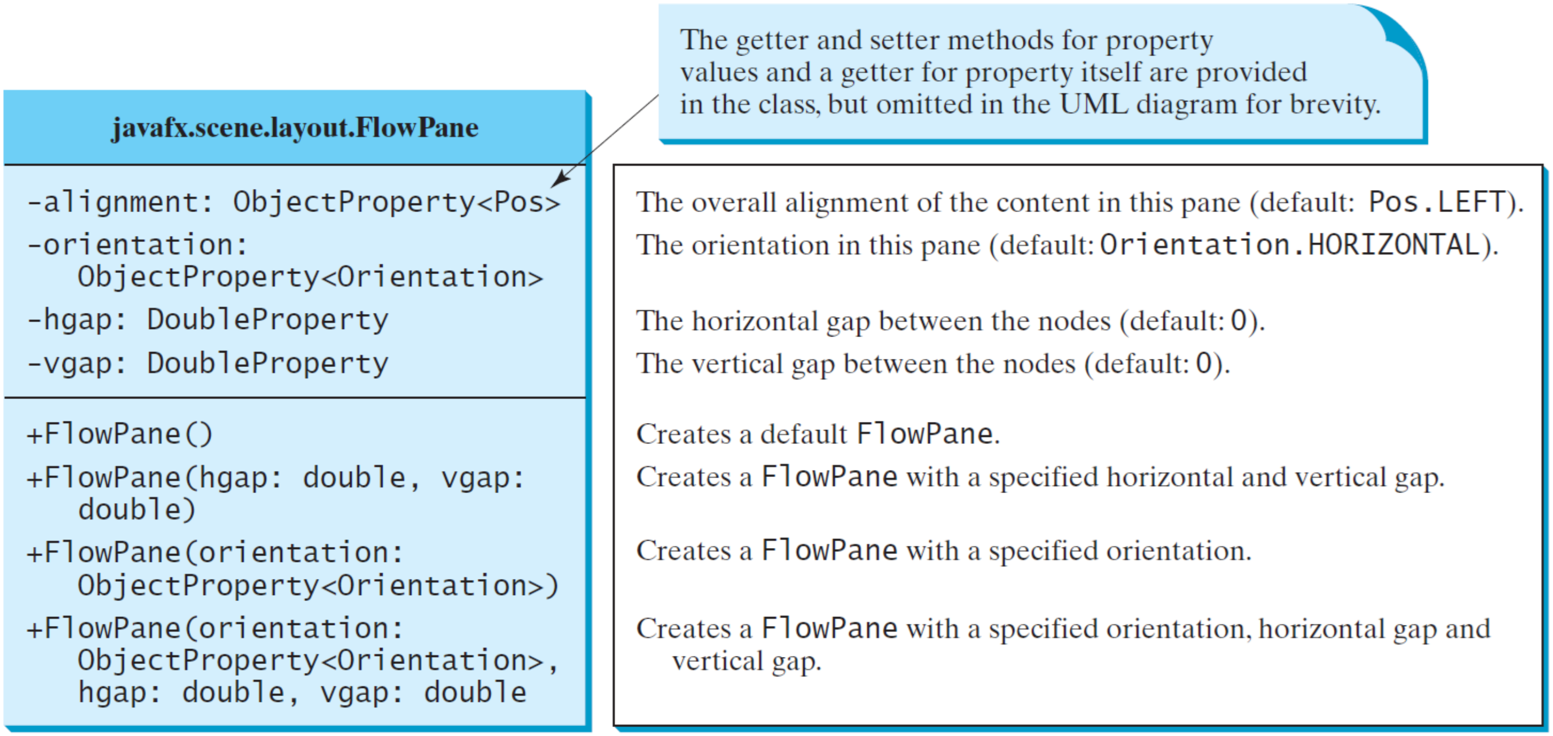
JavaFX has the following built-in layout panes



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.

FlowPane

- **lays out its children in a flow that wraps at the flowpane's boundary.**



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.layout.FlowPane

```
-alignment: ObjectProperty<Pos>
-orientation:
   ObjectProperty<Orientation>
-hgap: DoubleProperty
-vgap: DoubleProperty

+FlowPane()
+FlowPane(hgap: double, vgap:
   double)
+FlowPane(orientation:
   ObjectProperty<Orientation>)
+FlowPane(orientation:
   ObjectProperty<Orientation>,
   hgap: double, vgap: double
```

The overall alignment of the content in this pane (default: Pos.LEFT).
The orientation in this pane (default: Orientation.HORIZONTAL).

The horizontal gap between the nodes (default: 0).
The vertical gap between the nodes (default: 0).

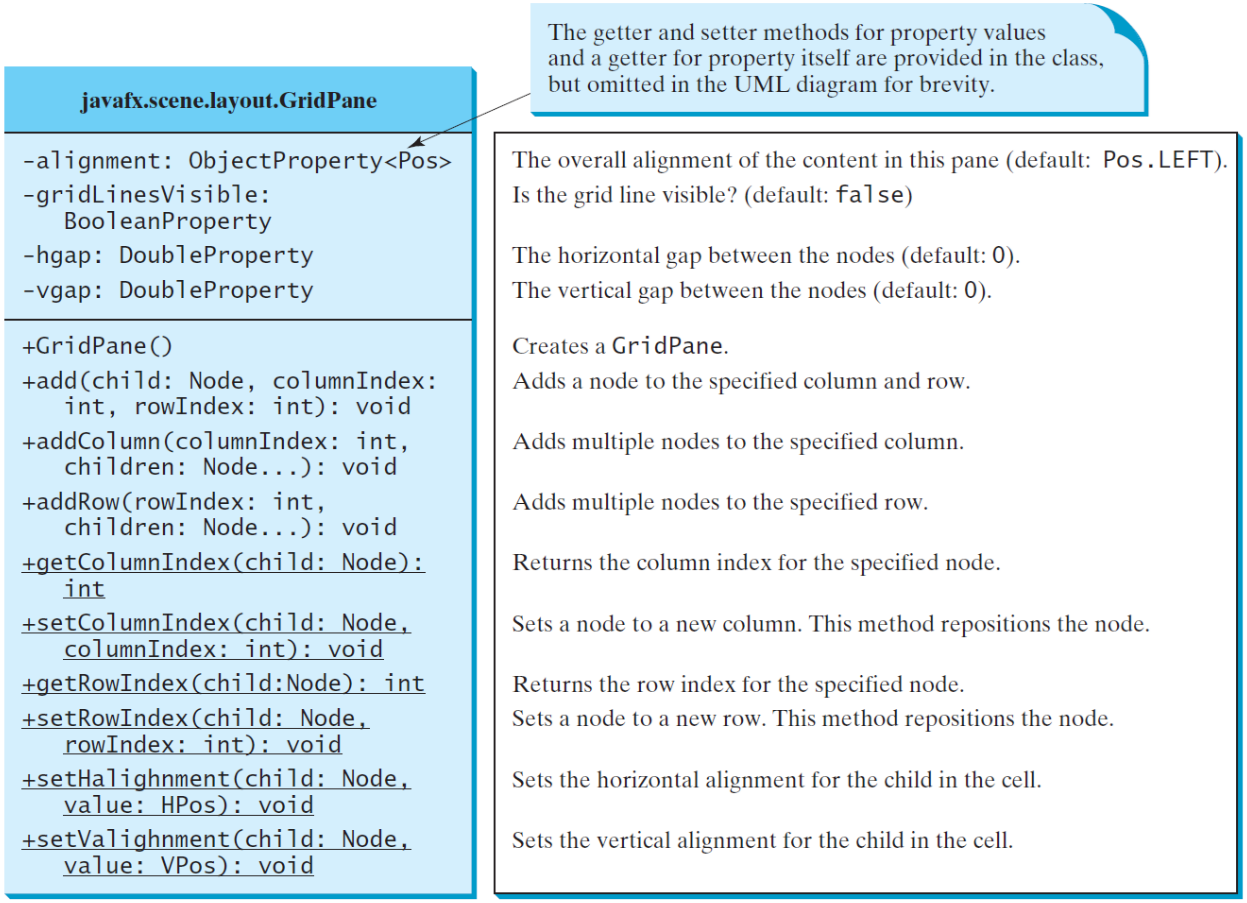
Creates a default FlowPane.
Creates a FlowPane with a specified horizontal and vertical gap.

Creates a FlowPane with a specified orientation.

Creates a FlowPane with a specified orientation, horizontal gap and vertical gap.

GridPane

Lays out its content nodes in some cell in Grid.



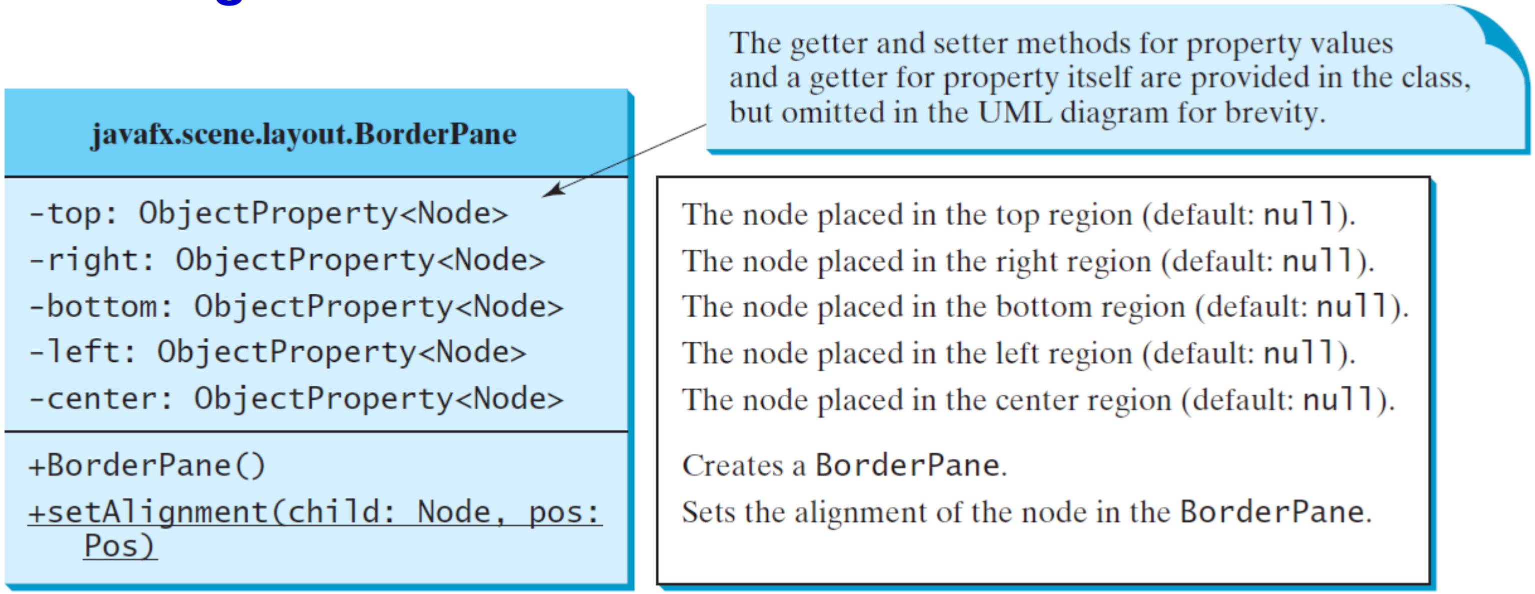
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.layout.GridPane

-alignment: ObjectProperty<Pos>	The overall alignment of the content in this pane (default: Pos.LEFT).
-gridLinesVisible: BooleanProperty	Is the grid line visible? (default: false)
-hgap: DoubleProperty	The horizontal gap between the nodes (default: 0).
-vgap: DoubleProperty	The vertical gap between the nodes (default: 0).
+GridPane()	Creates a GridPane.
+add(child: Node, columnIndex: int, rowIndex: int): void	Adds a node to the specified column and row.
+addColumn(columnIndex: int, children: Node...): void	Adds multiple nodes to the specified column.
+addRow(rowIndex: int, children: Node...): void	Adds multiple nodes to the specified row.
+getColumnIndex(child: Node): int	Returns the column index for the specified node.
+setColumnIndex(child: Node, columnIndex: int): void	Sets a node to a new column. This method repositions the node.
+getRowIndex(child:Node): int	Returns the row index for the specified node.
+setRowIndex(child: Node, rowIndex: int): void	Sets a node to a new row. This method repositions the node.
+setHalighnment(child: Node, value: HPos): void	Sets the horizontal alignment for the child in the cell.
+setValighnment(child: Node, value: VPos): void	Sets the vertical alignment for the child in the cell.

BorderPane

Lays out its content nodes in the top, bottom, right, left, or center region.



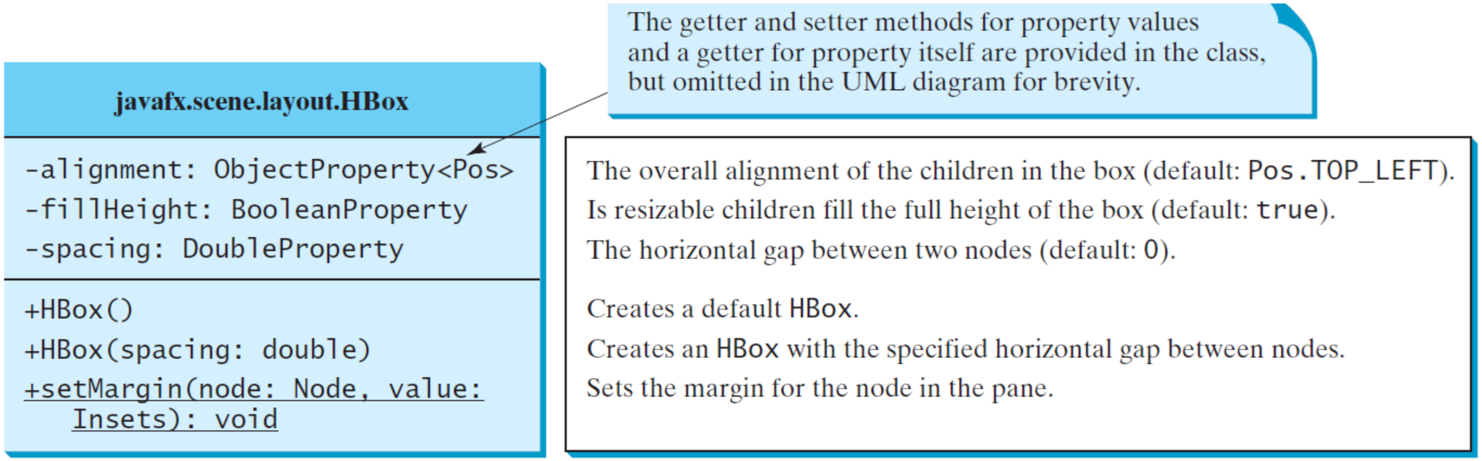
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.layout.BorderPane

-top: ObjectProperty<Node>	The node placed in the top region (default: null).
-right: ObjectProperty<Node>	The node placed in the right region (default: null).
-bottom: ObjectProperty<Node>	The node placed in the bottom region (default: null).
-left: ObjectProperty<Node>	The node placed in the left region (default: null).
-center: ObjectProperty<Node>	The node placed in the center region (default: null).
+BorderPane()	Creates a BorderPane.
+setAlignment(child: Node, pos: Pos)	Sets the alignment of the node in the BorderPane.

HBox

Arranges its content nodes horizontally in a single row.



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.layout.HBox

-alignment: ObjectProperty<Pos>
-fillHeight: BooleanProperty
-spacing: DoubleProperty

+HBox()
+HBox(spacing: double)
+setMargin(node: Node, value:
   Insets): void
```

The overall alignment of the children in the box (default: Pos.TOP_LEFT).

Is resizable children fill the full height of the box (default: true).

The horizontal gap between two nodes (default: 0).

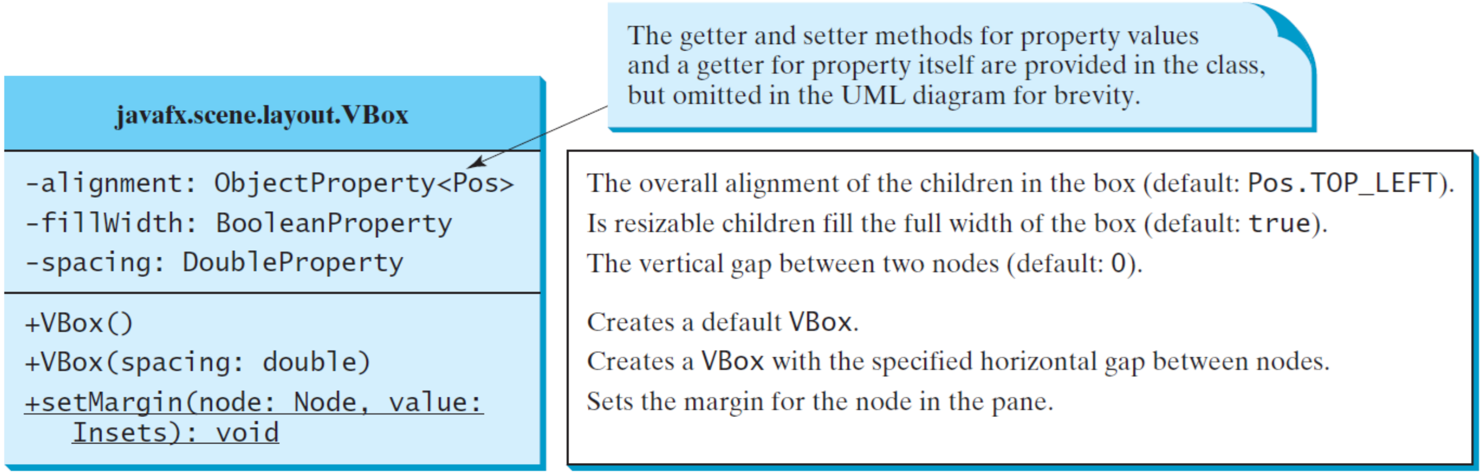
Creates a default HBox.

Creates an HBox with the specified horizontal gap between nodes.

Sets the margin for the node in the pane.

- □ **Insets: Padding -> Insets(top, right, bottom, left)**

VBox



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.layout.VBox

```
-alignment: ObjectProperty<Pos>
-fillWidth: BooleanProperty
-spacing: DoubleProperty

+VBox()
+VBox(spacing: double)
+setMargin(node: Node, value:
   Insets): void
```

The overall alignment of the children in the box (default: `Pos.TOP_LEFT`).

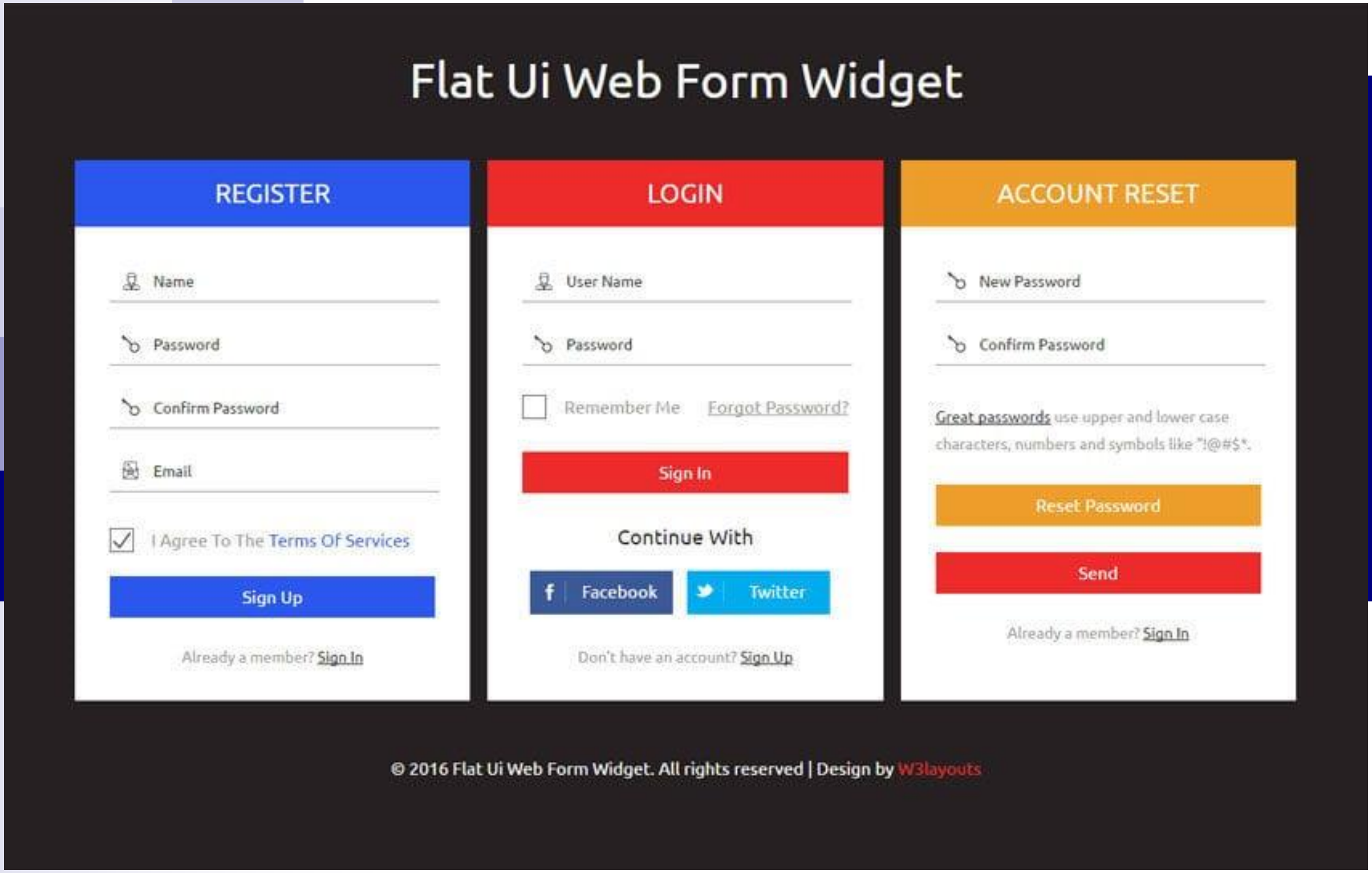
Is resizable children fill the full width of the box (default: `true`).

The vertical gap between two nodes (default: `0`).

Creates a default `VBox`.

Creates a `VBox` with the specified horizontal gap between nodes.

Sets the margin for the node in the pane.



Flat Ui Web Form Widget

Already a member? Sign In

Don't have an account? Sign Up

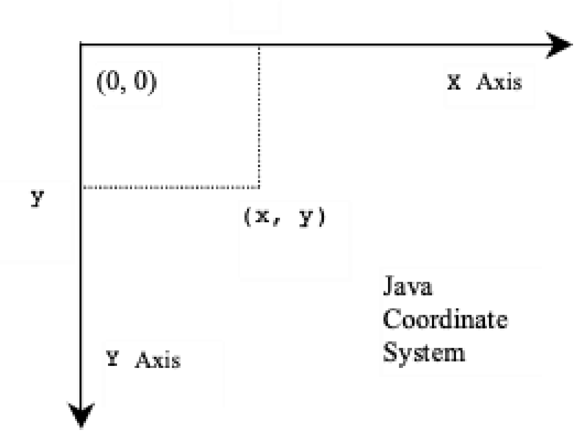
Great passwords use upper and lower case characters, numbers and symbols like "!@#$*.

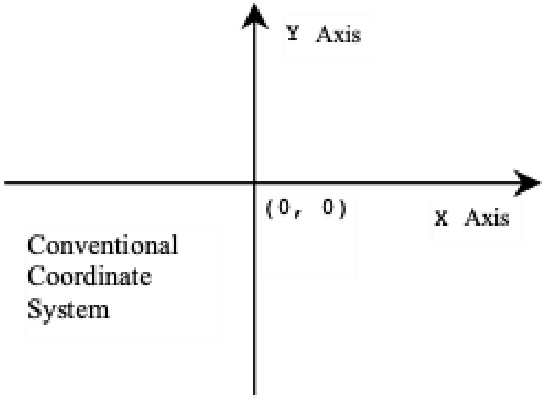
Already a member? Sign In

© 2016 Flat Ui Web Form Widget. All rights reserved | Design by W3layouts

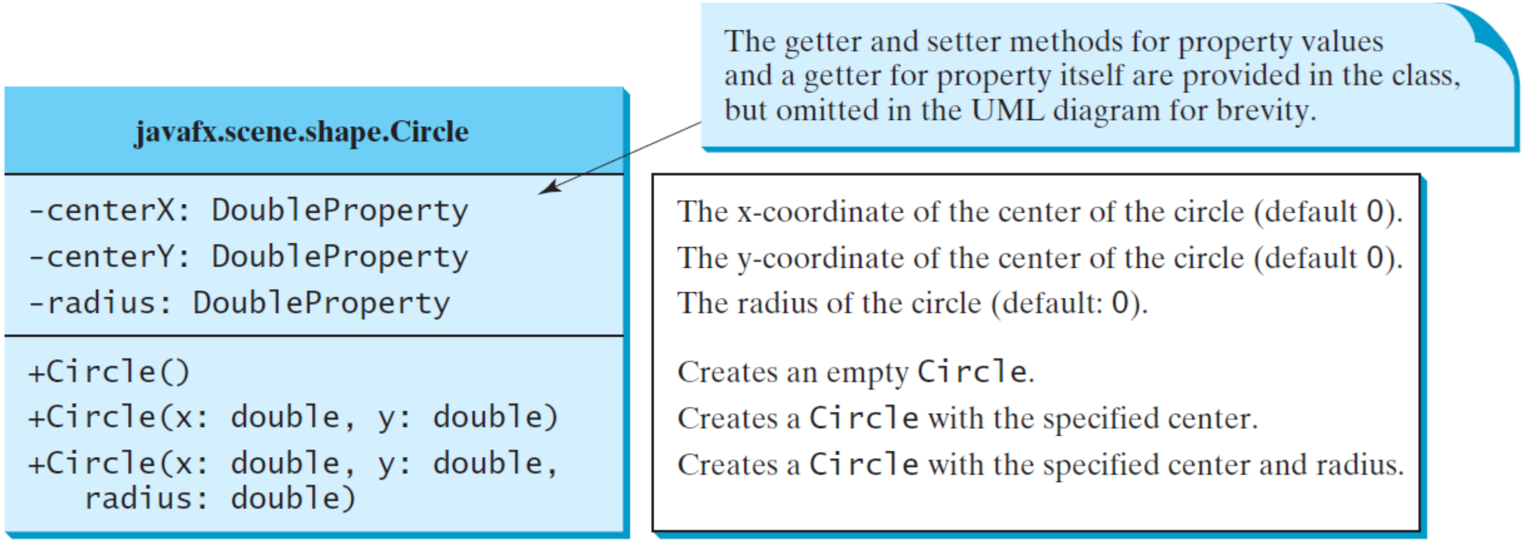
Display a Shape

This example displays a circle in the center of the pane.





Circle



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.Circle

```
-centerX: DoubleProperty
-centerY: DoubleProperty
-radius: DoubleProperty

+Circle()
+Circle(x: double, y: double)
+Circle(x: double, y: double,
   radius: double)
```

The x-coordinate of the center of the circle (default 0).

The y-coordinate of the center of the circle (default 0).

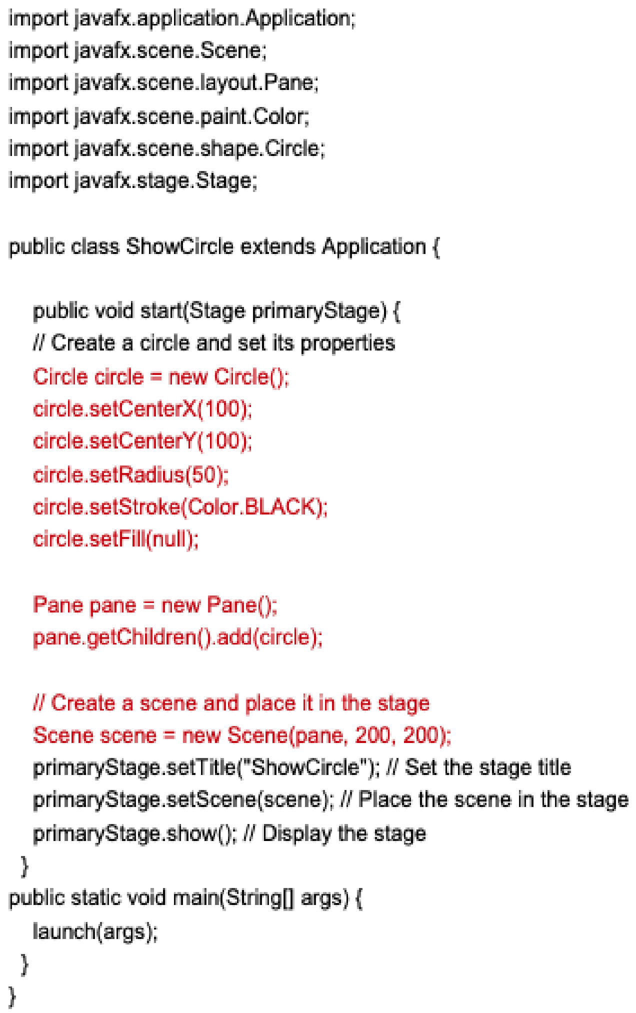
The radius of the circle (default: 0).

Creates an empty Circle.

Creates a Circle with the specified center.

Creates a Circle with the specified center and radius.

Display a Shape



```
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 {

    public void start(Stage primaryStage) {
    // Create a circle and set its properties
    Circle circle = new Circle();
    circle.setCenterX(100);
    circle.setCenterY(100);
    circle.setRadius(50);
    circle.setStroke(Color.BLACK);
    circle.setFill(null);

    Pane pane = new Pane();
    pane.getChildren().add(circle);

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 200, 200);
    primaryStage.setTitle("ShowCircle"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
public static void main(String[] args) {
    launch(args);
  }
}
```

Binding Properties

JavaFX introduces a new concept called binding property that enables a **target object to be bound to a source object**.

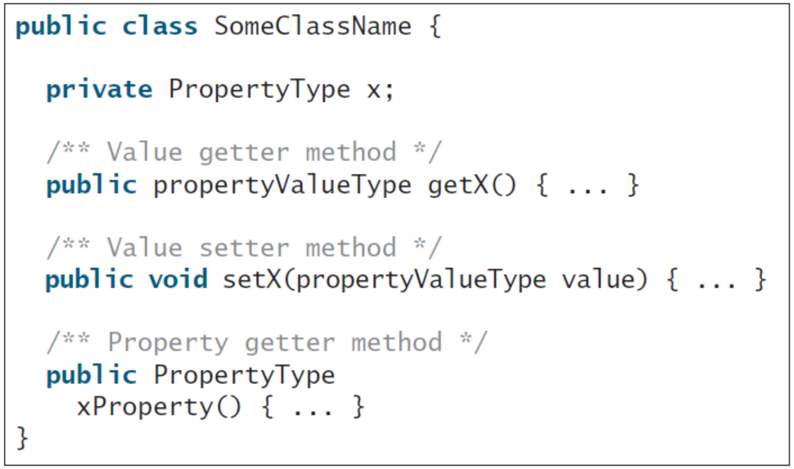
If the value in the source object changes, the target property is also changed automatically

The target object is simply called a *binding object* or a *binding property*.

Bound Circle

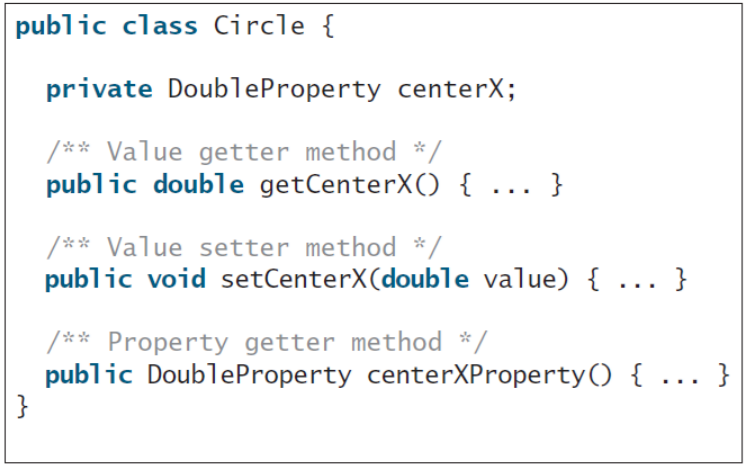
```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;

public class ShowCircleCentered extends Application {
  @Override // Override the start method in the Application class
  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); // Add circle to the pane

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

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

Binding Property:
getter, setter, and property getter



```
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() { ... }
}
```

(a) x is a binding property



```
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() { ... }
}
```

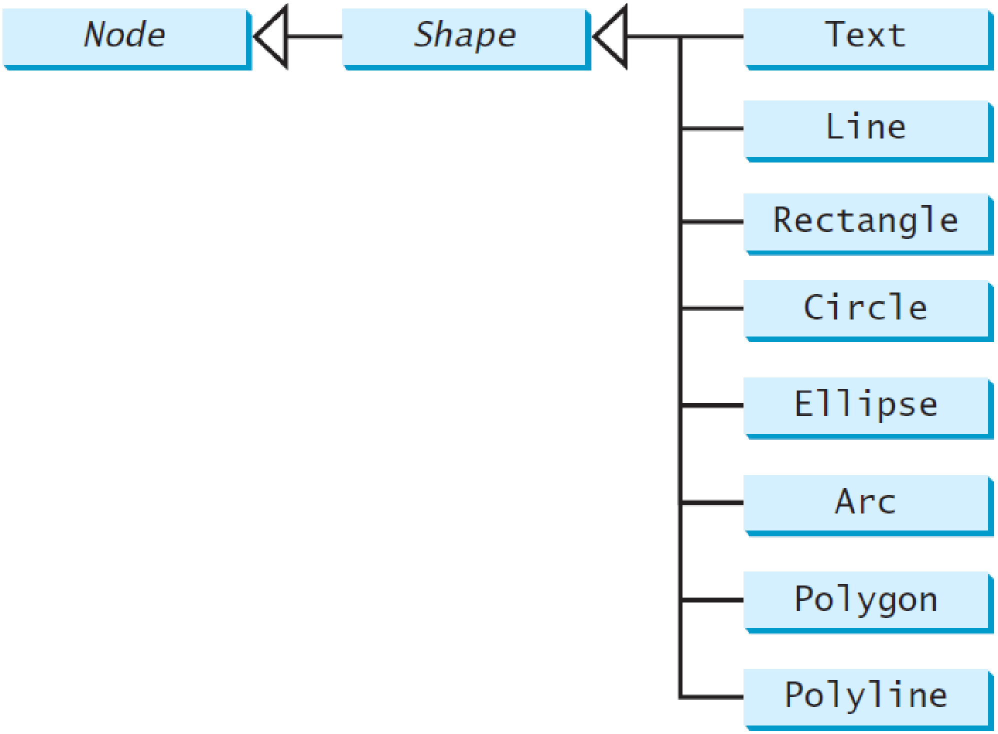
(b) centerX is binding property

It can happen both ways: Bidirectional Binding

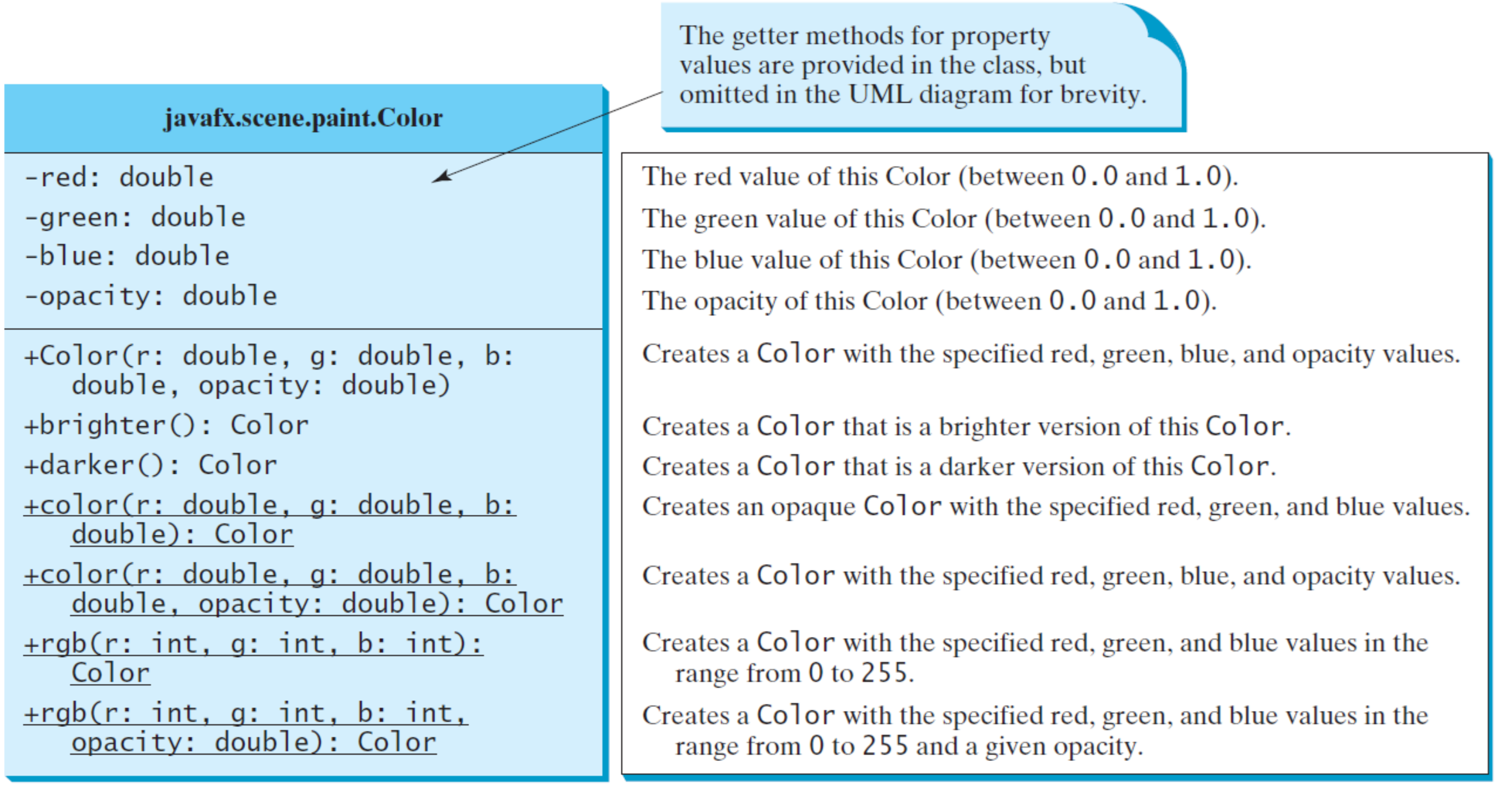
```
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;

public class BidirectionalBindingDemo {
  public static void main(String[] args) {
    DoubleProperty d1 = new SimpleDoubleProperty(1);
    DoubleProperty d2 = new SimpleDoubleProperty(2);
    d1.bind(d2);
    System.out.println("d1 is " + d1.getValue()
      + " and d2 is " + d2.getValue());
    d1.setValue(50.1);
    System.out.println("d1 is " + d1.getValue()
      + " and d2 is " + d2.getValue());
    d2.setValue(70.2);
    System.out.println("d1 is " + d1.getValue()
      + " and d2 is " + d2.getValue());
  }
}
```

Shapes

JavaFX provides many shape classes for drawing texts, lines, circles, rectangles, ellipses, arcs, polygons, and polylines.



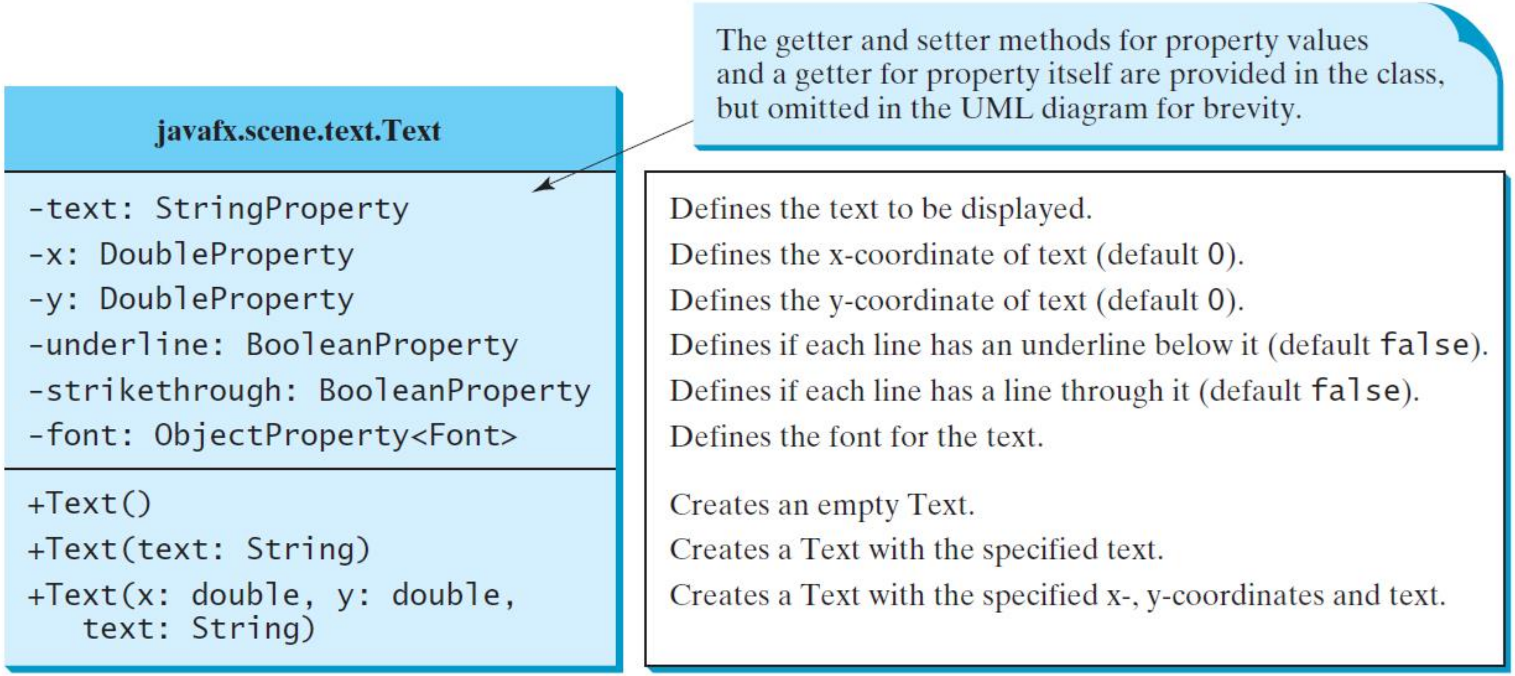
The Color Class



The getter methods for property values are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.paint.Color	
-red: double	The red value of this Color (between 0.0 and 1.0).
-green: double	The green value of this Color (between 0.0 and 1.0).
-blue: double	The blue value of this Color (between 0.0 and 1.0).
-opacity: double	The opacity of this Color (between 0.0 and 1.0).
+Color(r: double, g: double, b: double, opacity: double)	Creates a Color with the specified red, green, blue, and opacity values.
+brighter(): Color	Creates a Color that is a brighter version of this Color.
+darker(): Color	Creates a Color that is a darker version of this Color.
+color(r: double, g: double, b: double): Color	Creates an opaque Color with the specified red, green, and blue values.
+color(r: double, g: double, b: double, opacity: double): Color	Creates a Color with the specified red, green, blue, and opacity values.
+rgb(r: int, g: int, b: int): Color	Creates a Color with the specified red, green, and blue values in the range from 0 to 255.
+rgb(r: int, g: int, b: int, opacity: double): Color	Creates a Color with the specified red, green, and blue values in the range from 0 to 255 and a given opacity.

Text



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.text.Text

```
-text: StringProperty
-x: DoubleProperty
-y: DoubleProperty
-underline: BooleanProperty
-strikethrough: BooleanProperty
-font: ObjectProperty<Font>
```

Defines the text to be displayed.

Defines the x-coordinate of text (default 0).

Defines the y-coordinate of text (default 0).

Defines if each line has an underline below it (default `false`).

Defines if each line has a line through it (default `false`).

Defines the font for the text.

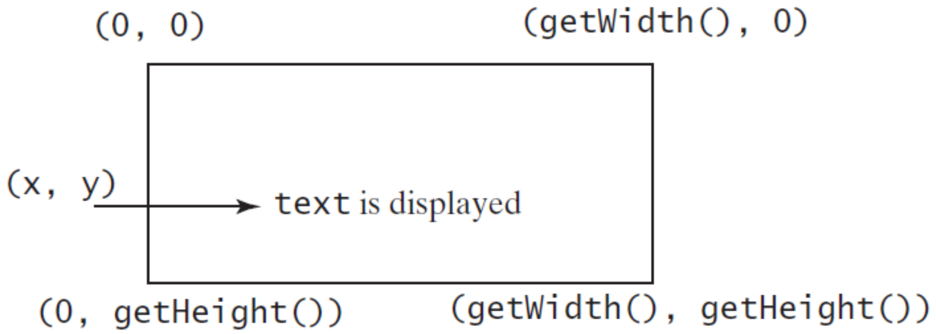
```
+Text()
+Text(text: String)
+Text(x: double, y: double,
   text: String)
```

Creates an empty Text.

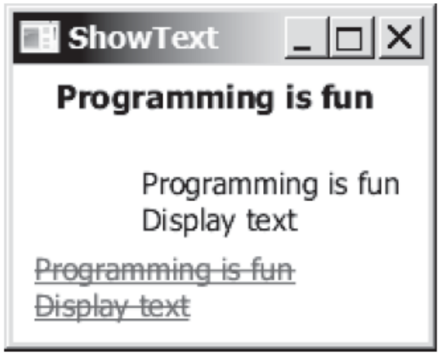
Creates a Text with the specified text.

Creates a Text with the specified x-, y-coordinates and text.

Text Example



(a) Text(x, y, text)



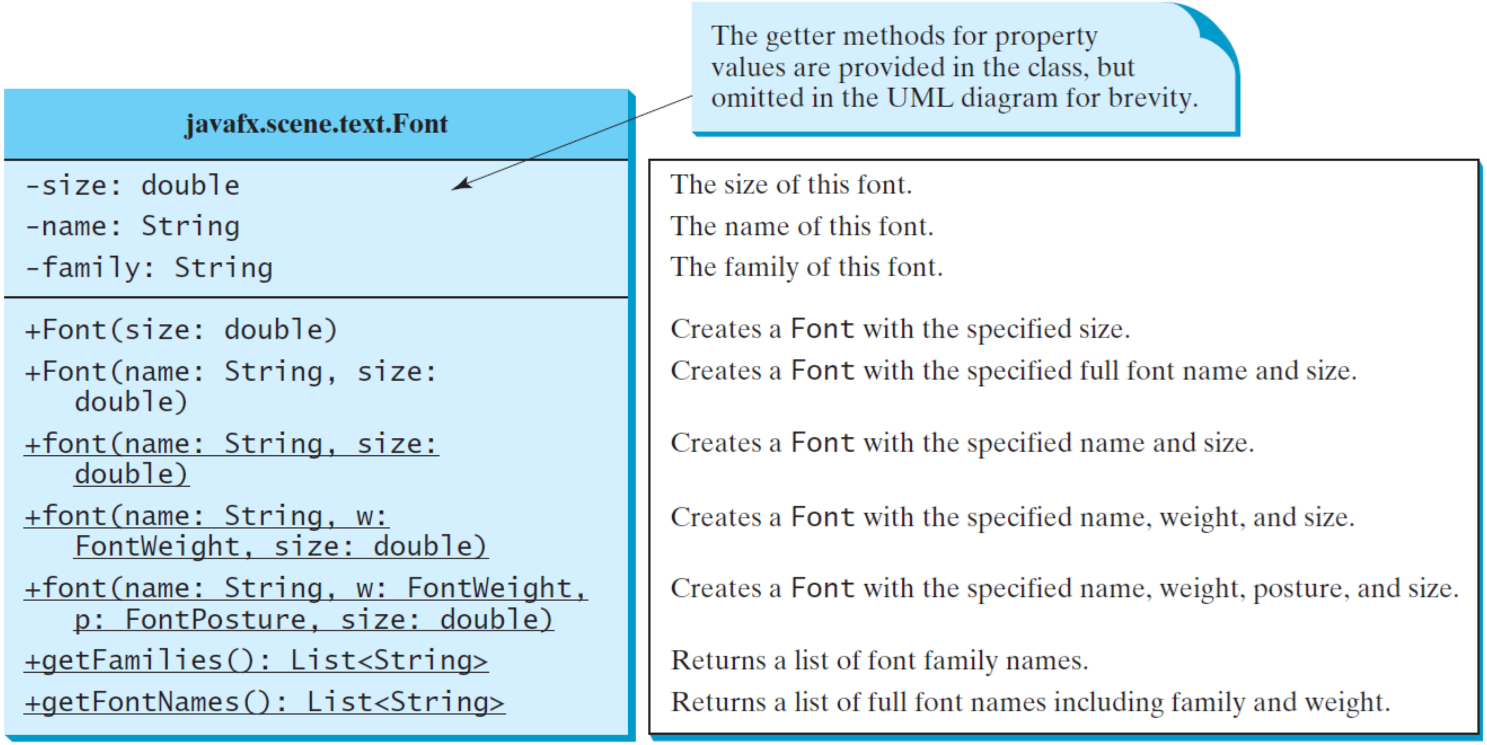
Programming is fun

Programming is fun
Display text

~~Programming is fun~~
~~Display text~~

(b) *Three* Text *objects are displayed*

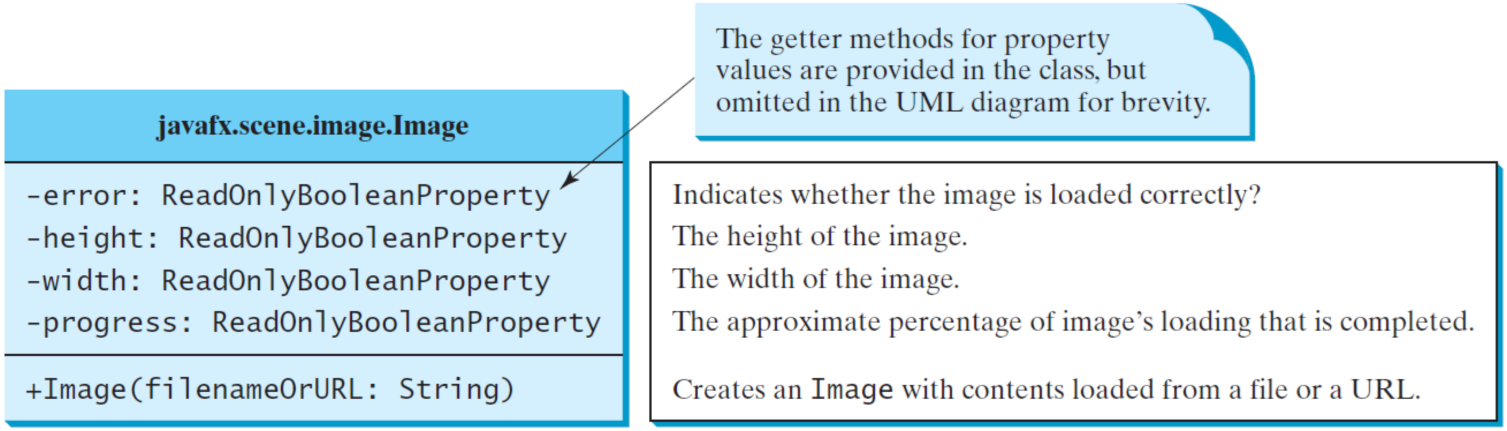
The Font Class



The getter methods for property values are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.text.Font	
-size: double	The size of this font.
-name: String	The name of this font.
-family: String	The family of this font.
+Font(size: double)	Creates a Font with the specified size.
+Font(name: String, size: double)	Creates a Font with the specified full font name and size.
+font(name: String, size: double)	Creates a Font with the specified name and size.
+font(name: String, w: FontWeight, size: double)	Creates a Font with the specified name, weight, and size.
+font(name: String, w: FontWeight, p: FontPosture, size: double)	Creates a Font with the specified name, weight, posture, and size.
+getFamilies(): List<String>	Returns a list of font family names.
+getFontNames(): List<String>	Returns a list of full font names including family and weight.

The Image Class



The getter methods for property values are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.image.Image

```
-error: ReadOnlyBooleanProperty
-height: ReadOnlyBooleanProperty
-width: ReadOnlyBooleanProperty
-progress: ReadOnlyBooleanProperty
```

```
+Image(filenameOrURL: String)
```

Indicates whether the image is loaded correctly?

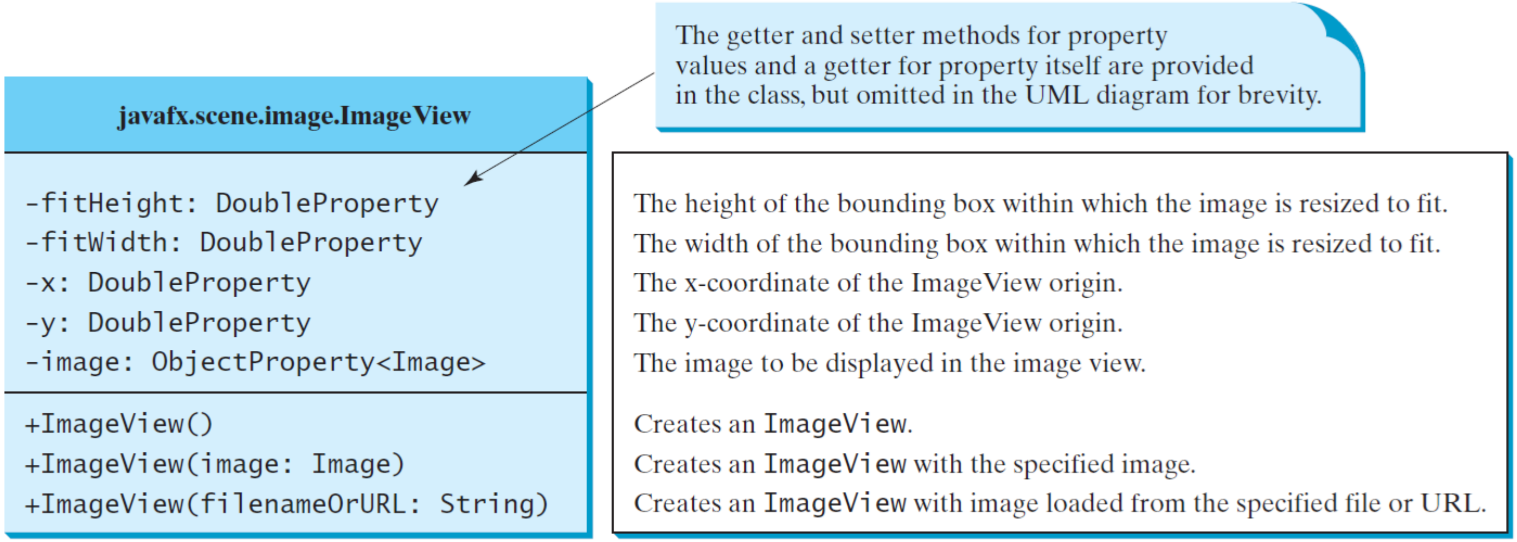
The height of the image.

The width of the image.

The approximate percentage of image’s loading that is completed.

Creates an `Image` with contents loaded from a file or a URL.

The ImageView Class



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.

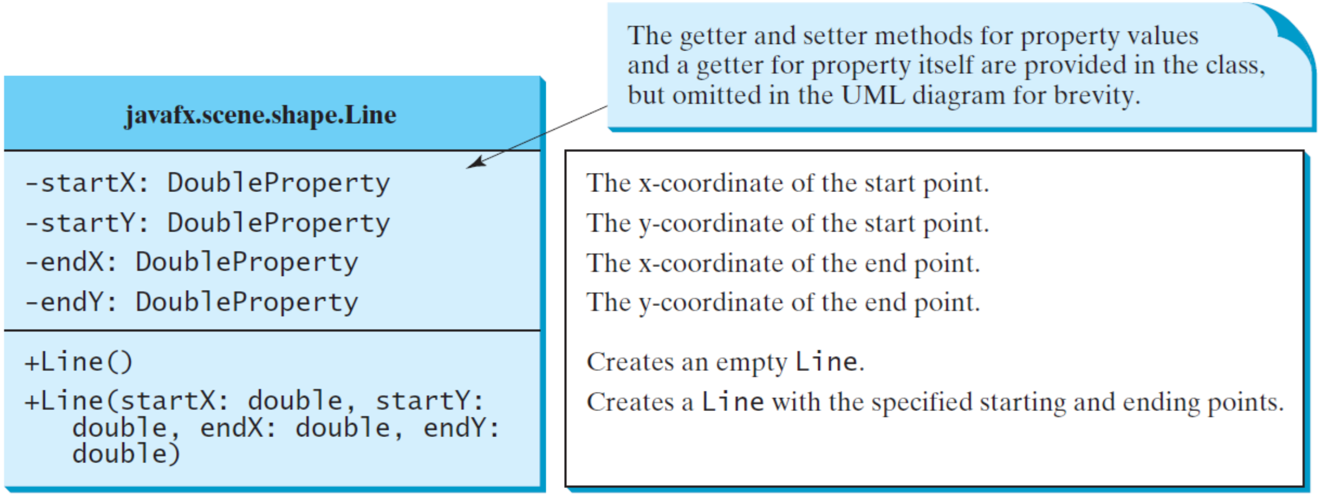
javafx.scene.image.ImageView

```
-fitHeight: DoubleProperty
-fitWidth: DoubleProperty
-x: DoubleProperty
-y: DoubleProperty
-image: ObjectProperty<Image>

+ImageView()
+ImageView(image: Image)
+ImageView(filenameOrURL: String)
```

The height of the bounding box within which the image is resized to fit.
The width of the bounding box within which the image is resized to fit.
The x-coordinate of the ImageView origin.
The y-coordinate of the ImageView origin.
The image to be displayed in the image view.

Creates an `ImageView`.
Creates an `ImageView` with the specified image.
Creates an `ImageView` with image loaded from the specified file or URL.

Line



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.Line

`-startX: DoubleProperty` The x-coordinate of the start point.

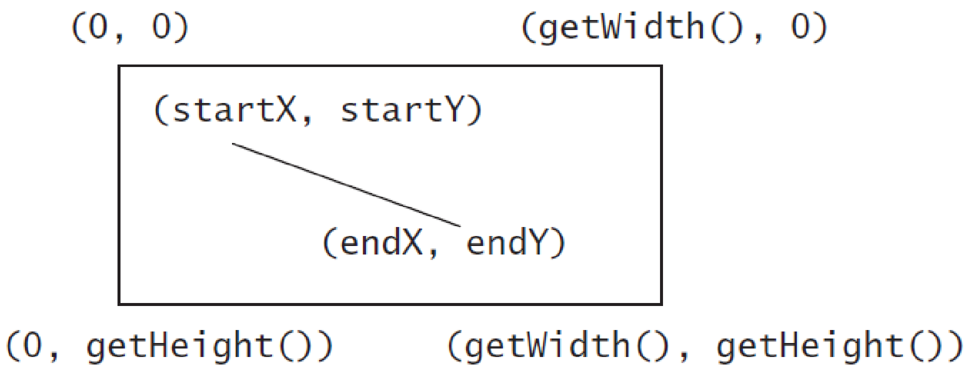
`-startY: DoubleProperty` The y-coordinate of the start point.

`-endX: DoubleProperty` The x-coordinate of the end point.

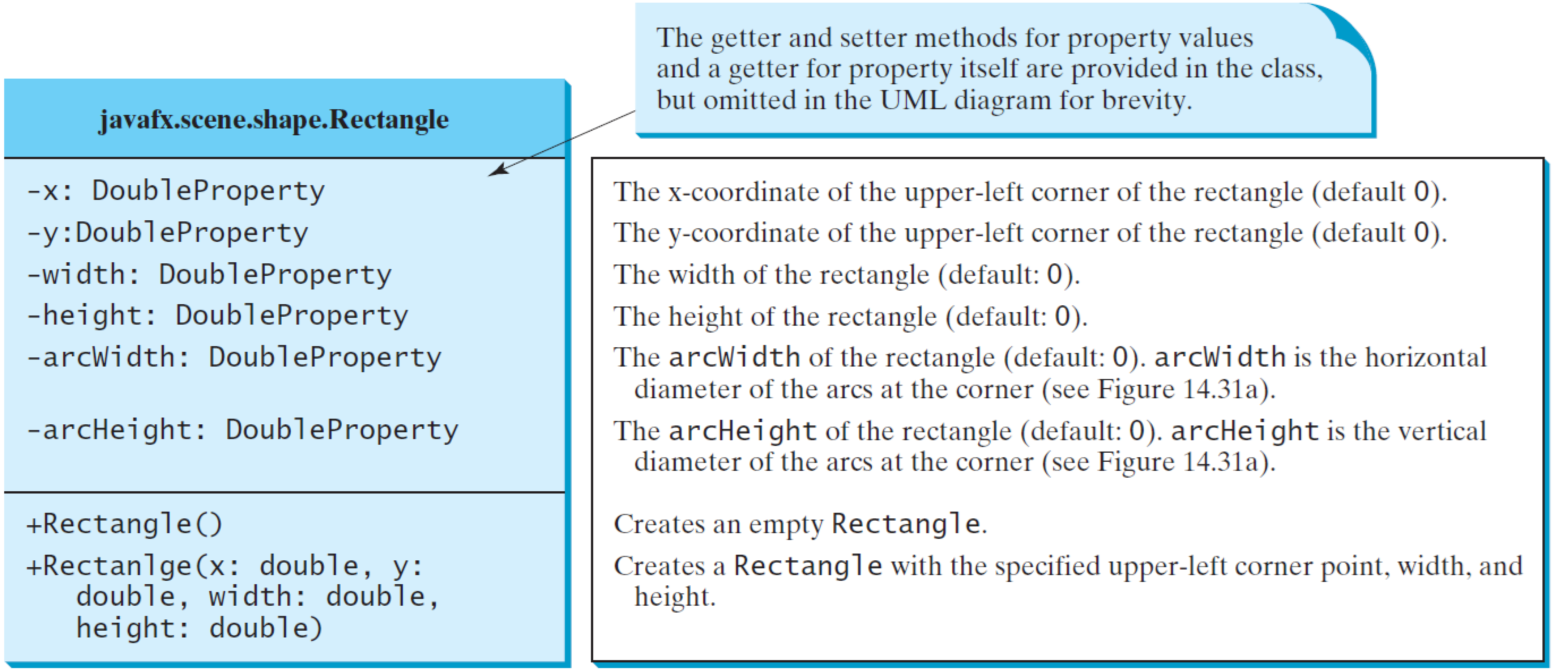
`-endY: DoubleProperty` The y-coordinate of the end point.

`+Line()` Creates an empty `Line`.

`+Line(startX: double, startY: double, endX: double, endY: double)` Creates a `Line` with the specified starting and ending points.



Rectangle



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.Rectangle

```
-x: DoubleProperty
-y:DoubleProperty
-width: DoubleProperty
-height: DoubleProperty
-arcWidth: DoubleProperty

-arcHeight: DoubleProperty
```

```
+Rectangle()
+Rectanlge(x: double, y:
   double, width: double,
   height: double)
```

The x-coordinate of the upper-left corner of the rectangle (default 0).

The y-coordinate of the upper-left corner of the rectangle (default 0).

The width of the rectangle (default: 0).

The height of the rectangle (default: 0).

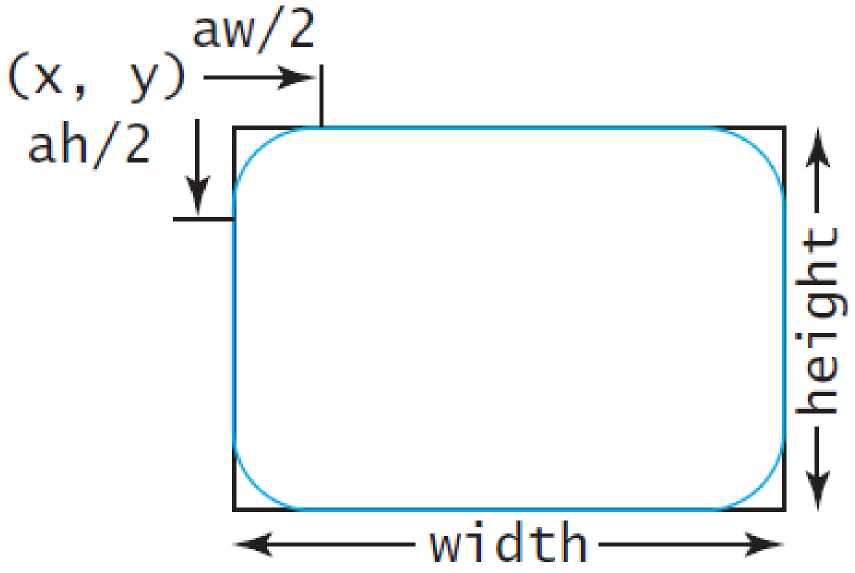
The `arcWidth` of the rectangle (default: 0). `arcWidth` is the horizontal diameter of the arcs at the corner (see Figure 14.31a).

The `arcHeight` of the rectangle (default: 0). `arcHeight` is the vertical diameter of the arcs at the corner (see Figure 14.31a).

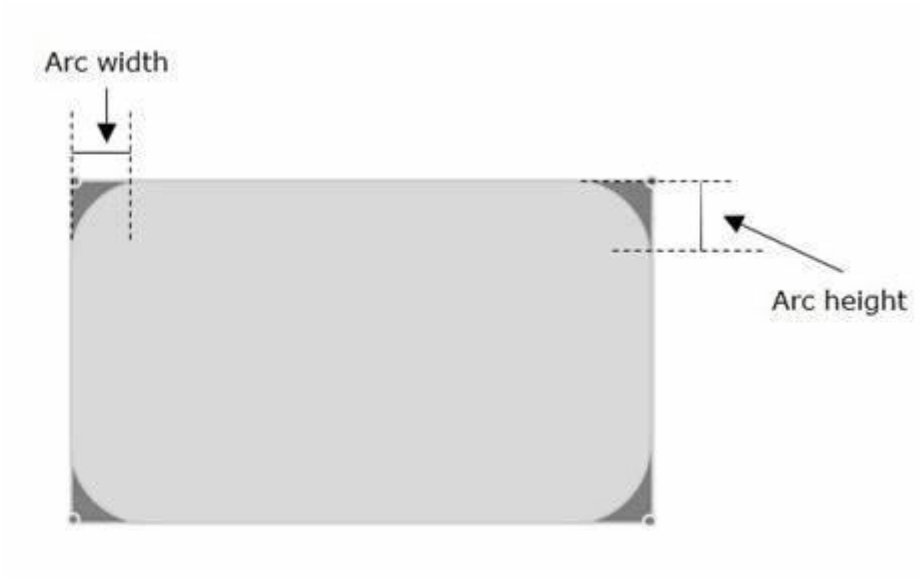
Creates an empty `Rectangle`.

Creates a `Rectangle` with the specified upper-left corner point, width, and height.

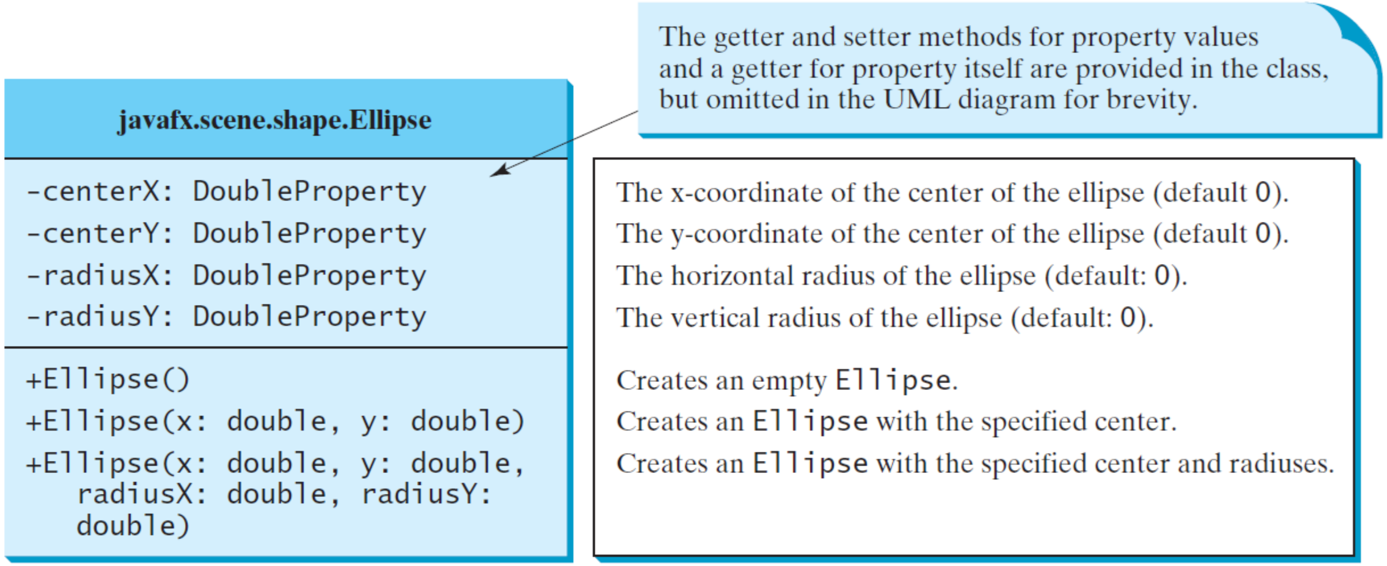
Rectangle Example



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



Ellipse



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.Ellipse

```
-centerX: DoubleProperty
-centerY: DoubleProperty
-radiusX: DoubleProperty
-radiusY: DoubleProperty

+Ellipse()
+Ellipse(x: double, y: double)
+Ellipse(x: double, y: double,
   radiusX: double, radiusY:
   double)
```

The x-coordinate of the center of the ellipse (default 0).

The y-coordinate of the center of the ellipse (default 0).

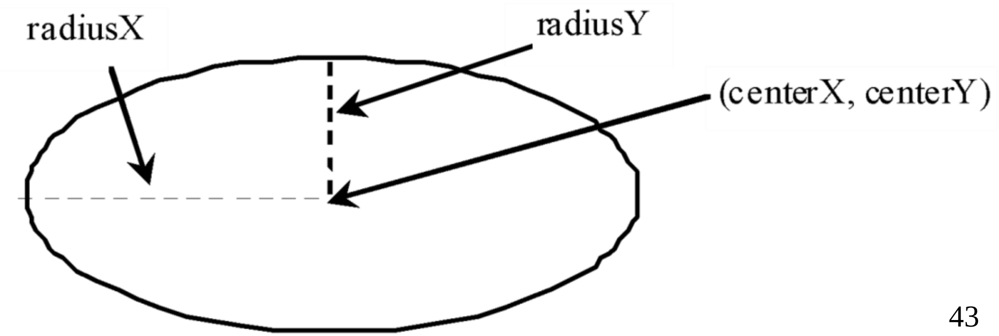
The horizontal radius of the ellipse (default: 0).

The vertical radius of the ellipse (default: 0).

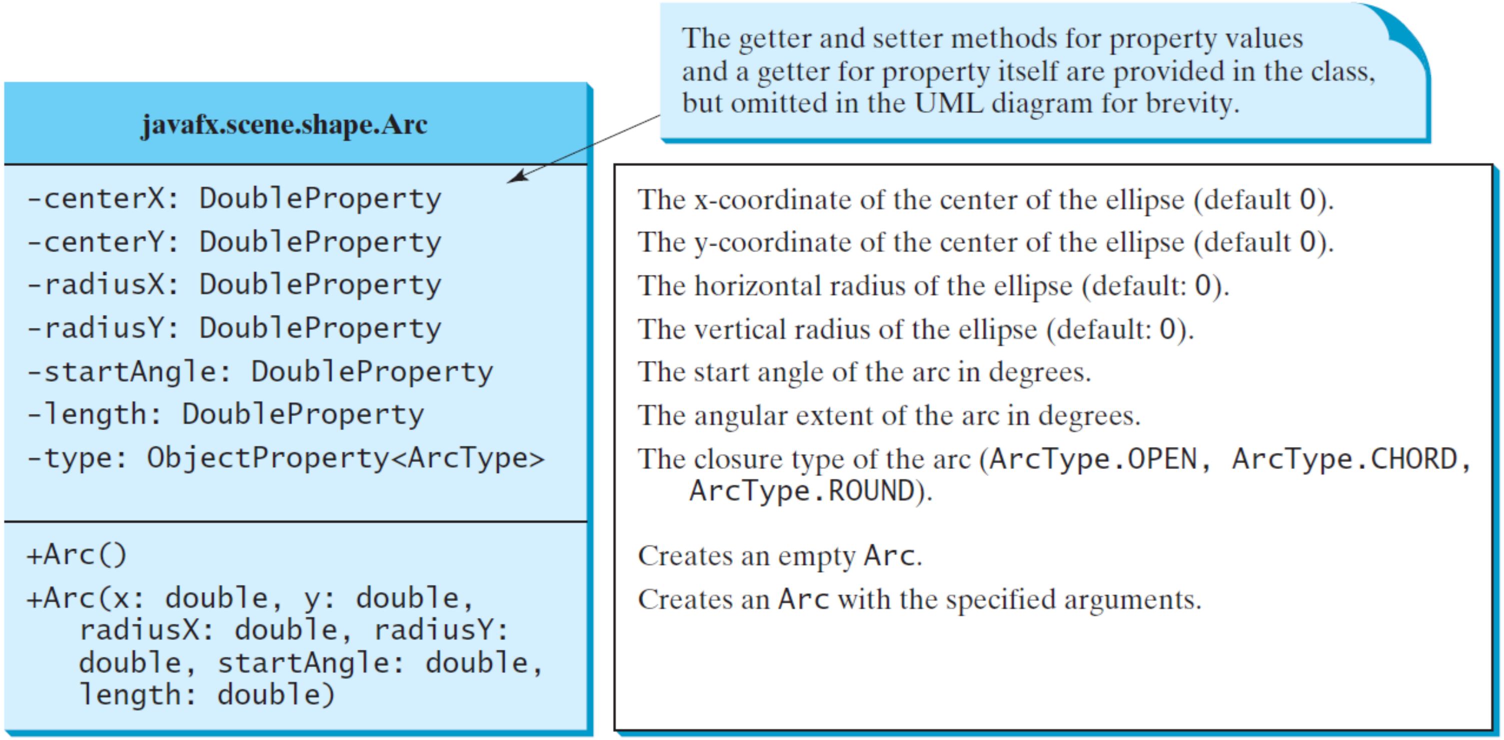
Creates an empty `Ellipse`.

Creates an `Ellipse` with the specified center.

Creates an `Ellipse` with the specified center and radiuses.



Arc

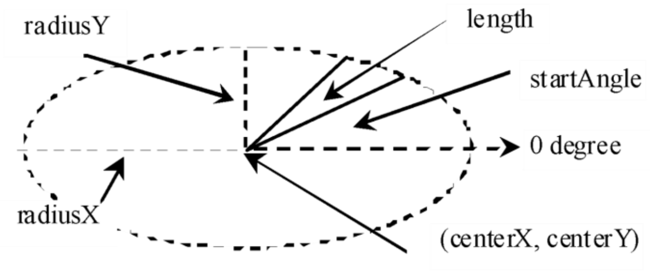


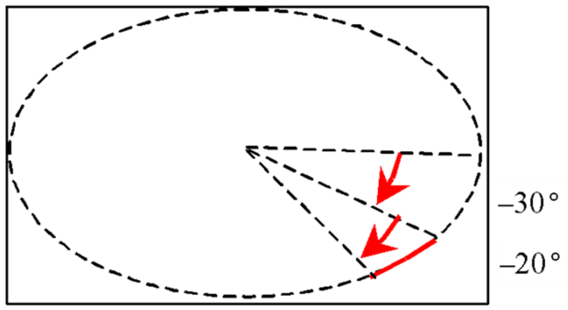
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.shape.Arc

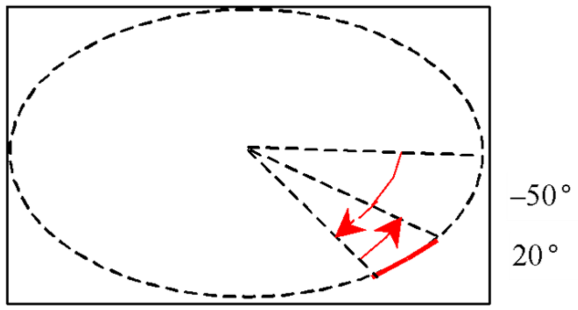
-centerX: DoubleProperty	The x-coordinate of the center of the ellipse (default 0).
-centerY: DoubleProperty	The y-coordinate of the center of the ellipse (default 0).
-radiusX: DoubleProperty	The horizontal radius of the ellipse (default: 0).
-radiusY: DoubleProperty	The vertical radius of the ellipse (default: 0).
-startAngle: DoubleProperty	The start angle of the arc in degrees.
-length: DoubleProperty	The angular extent of the arc in degrees.
-type: ObjectProperty<ArcType>	The closure type of the arc (ArcType.OPEN, ArcType.CHORD, ArcType.ROUND).
+Arc()	Creates an empty Arc.
+Arc(x: double, y: double, radiusX: double, radiusY: double, startAngle: double, length: double)	Creates an Arc with the specified arguments.

Arc Examples



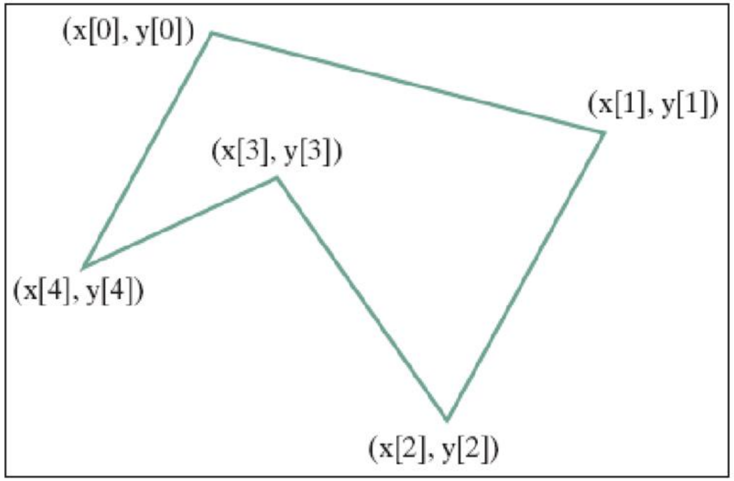


(a) Negative starting angle –30° and
negative spanning angle –20°

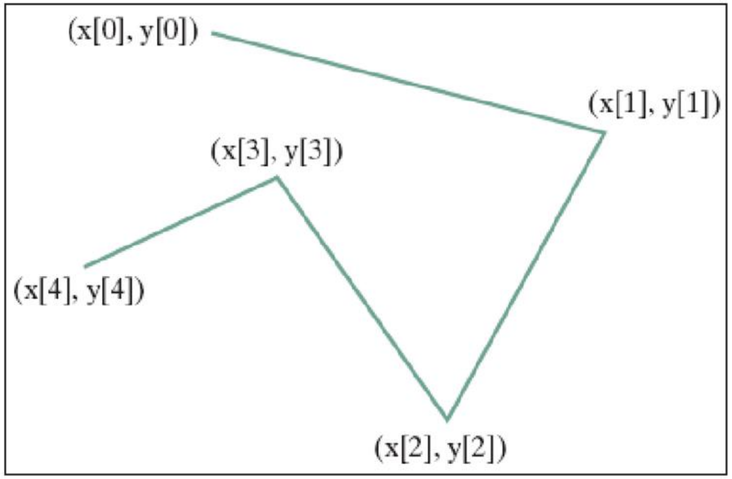


(b) Negative starting angle –50°
and positive spanning angle 20°

Polygon and Polyline

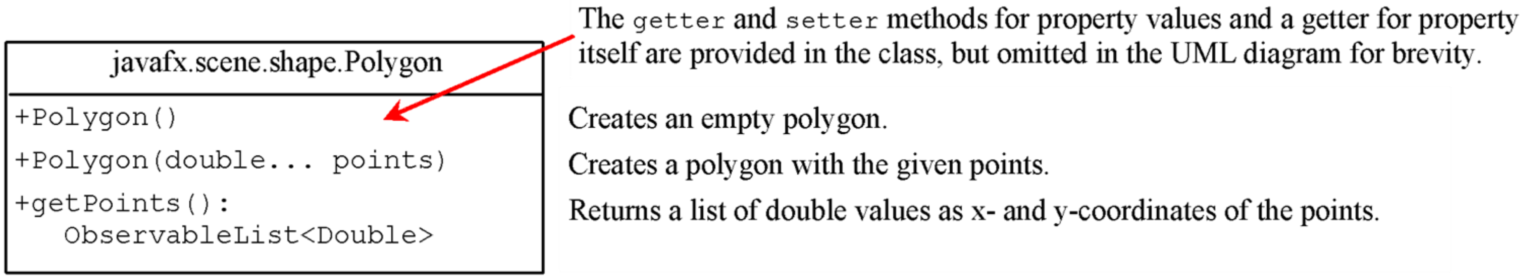


(a) Polygon



(b) Polyline

Polygon



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.Polygon

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

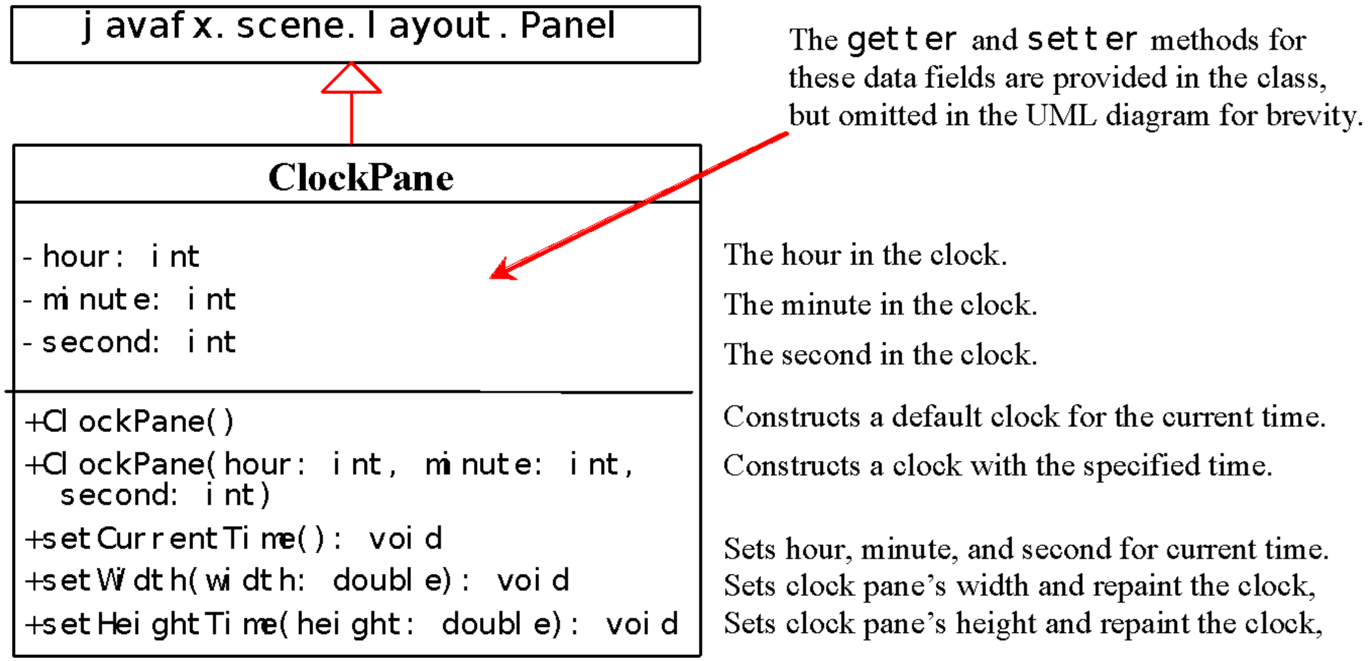
Creates an empty polygon.

Creates a polygon with the given points.

Returns a list of double values as x- and y-coordinates of the points.

Case Study: The ClockPane Class

This case study develops a class that displays a clock on a pane.



javafx.scene.layout.Panel

The getter and setter methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

ClockPane

- hour: int
- minute: int
- second: int

+ClockPane()
+ClockPane(hour: int, minute: int, second: int)
+setCurrentTime(): void
+setWidth(width: double): void
+setHeightTime(height: double): void

The hour in the clock.
The minute in the clock.
The second in the clock.

Constructs a default clock for the current time.
Constructs a clock with the specified time.

Sets hour, minute, and second for current time.
Sets clock pane's width and repaint the clock,
Sets clock pane's height and repaint the clock,

Use the ClockPane Class DispalyClock as Controller