Chapter 31 Advanced JavaFX

Objectives

- To specify styles for UI nodes using JavaFX CSS (§31.2).
- To create quadratic curve, cubic curve, and path using the QuadCurve, CubicCurve, and Path classes (§31.3).
- To translation, rotation, and scaling to perform coordinate transformations for nodes (§31.4).
- To define a shape’s border using various types of strokes (§31.5).
- To create menus using the Menu, MenuItem, CheckMenuItem, and RadioMemuItem classes (§31.6).
- To create context menus using the ContextMenu class (§31.7).
- To use SplitPane to create adjustable horizontal and vertical panes (§31.8).
- To create tab panes using the TabPane control (§31.9).
- To create and display tables using the TableView and TableColumn classes (§31.10).
- To create JavaFX user interfaces using FMXL and the visual Scene Builder (§31.11).

JavaFX CSS

A JavaFX style property is defined with a prefix **–fx-** to distinquish it from a property in CSS. All the available JavaFX properties are defined in http://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html. Listing 31.1 gives an example of a style sheet .

Style Class and Style ID

A style sheet uses the style class or style id to define styles. Mutiple style classes can be applied to a single node and a style id to a unique node. The syntax **.styleclass** defines a style class. The syntax **#styleid** defines a style id.

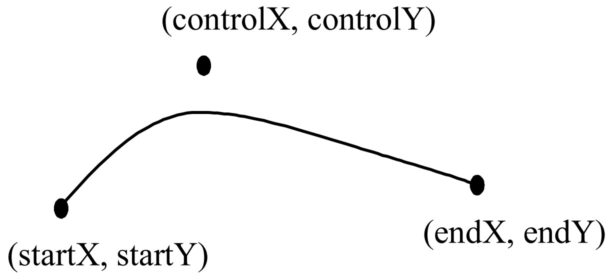
mystyle.css

StyleSheetDemo

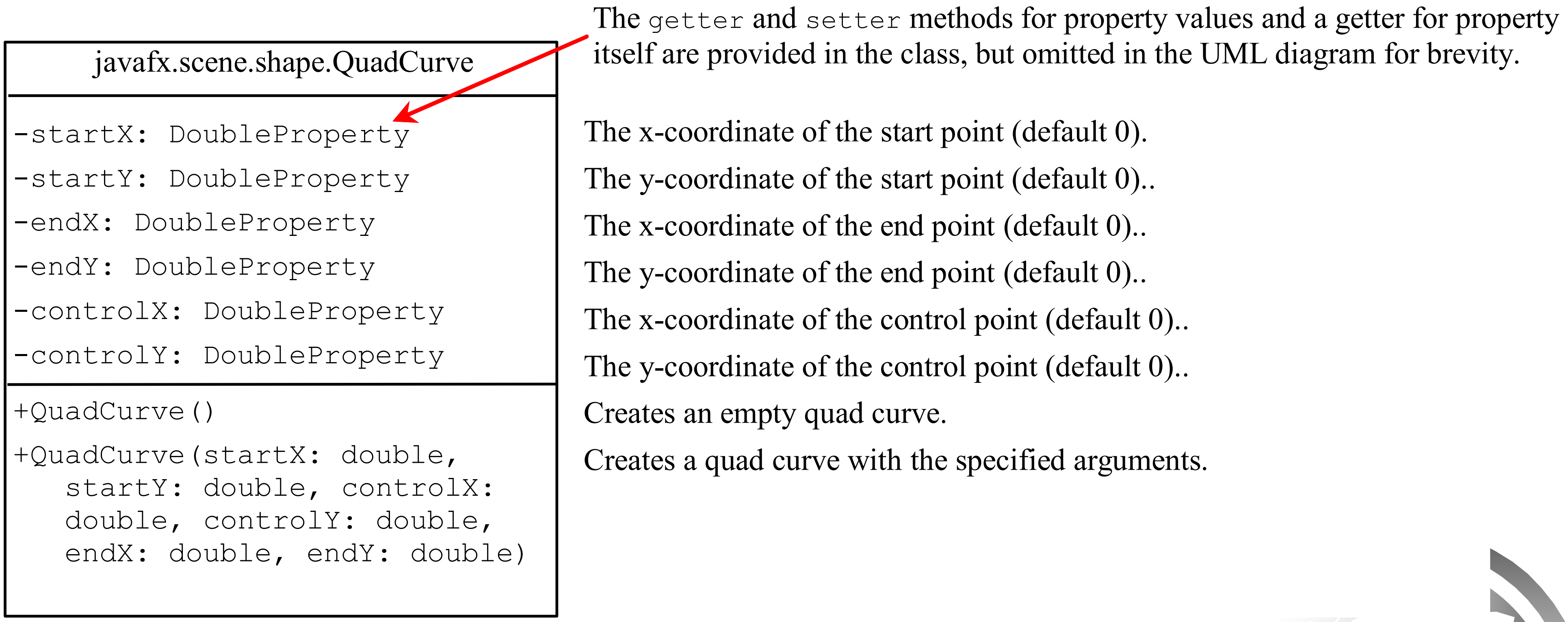
QuadCurve

A quadratic curve is mathematically defined as a quadratic polynomial. To create a **QuadCurve**, use its no-arg constructor or the following constructor:

QuadCurve(**double** startX, **double** startY,
 double controlX, **double** controlY, **double** endX, **double** endY)



QuadCurve



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.

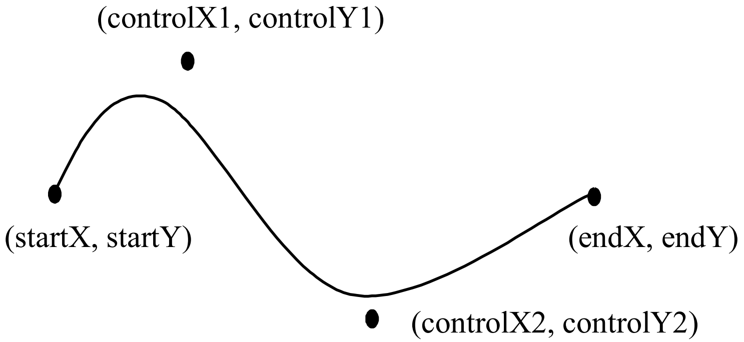
javafx.scene.shape.QuadCurve

`-startX: DoubleProperty`	The x-coordinate of the start point (default 0).
`-startY: DoubleProperty`	The y-coordinate of the start point (default 0)..
`-endX: DoubleProperty`	The x-coordinate of the end point (default 0)..
`-endY: DoubleProperty`	The y-coordinate of the end point (default 0)..
`-controlX: DoubleProperty`	The x-coordinate of the control point (default 0)..
`-controlY: DoubleProperty`	The y-coordinate of the control point (default 0)..
`+QuadCurve()`	Creates an empty quad curve.
`+QuadCurve(startX: double, startY: double, controlX: double, controlY: double, endX: double, endY: double)`	Creates a quad curve with the specified arguments.

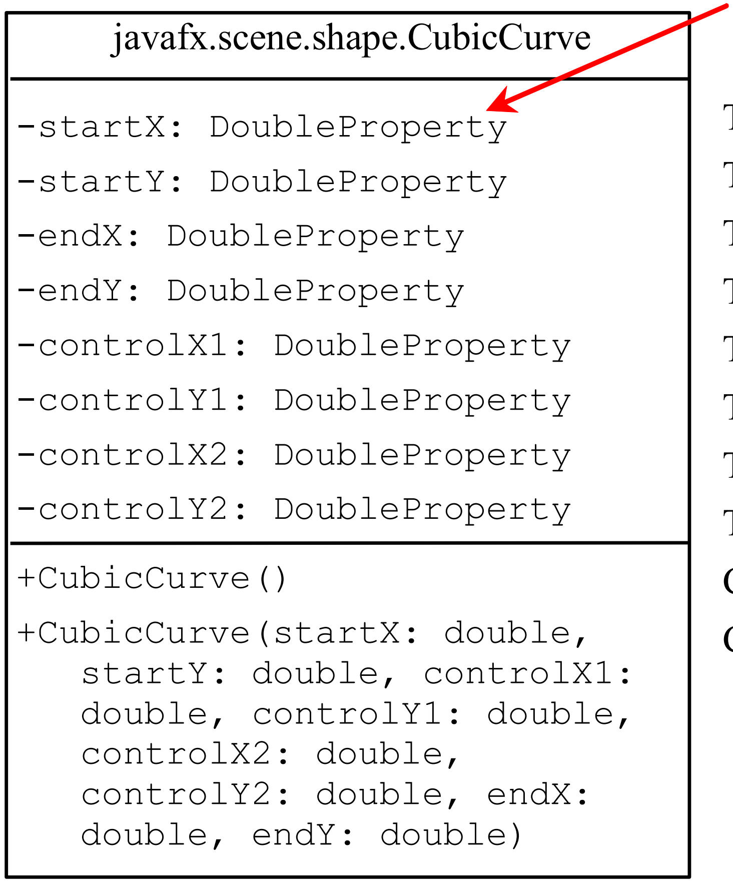
CubicCurve

A cubic curve is mathematically defined as a cubic polynomial. To create a **CubicCurve**, use its no-arg constructor or the following constructor:

CubicCurve(**double** startX, **double** startY, **double** controlX1,
double controlY1, **double** controlX2, **double** controlY2,
double endX, **double** endY)



CubicCurve

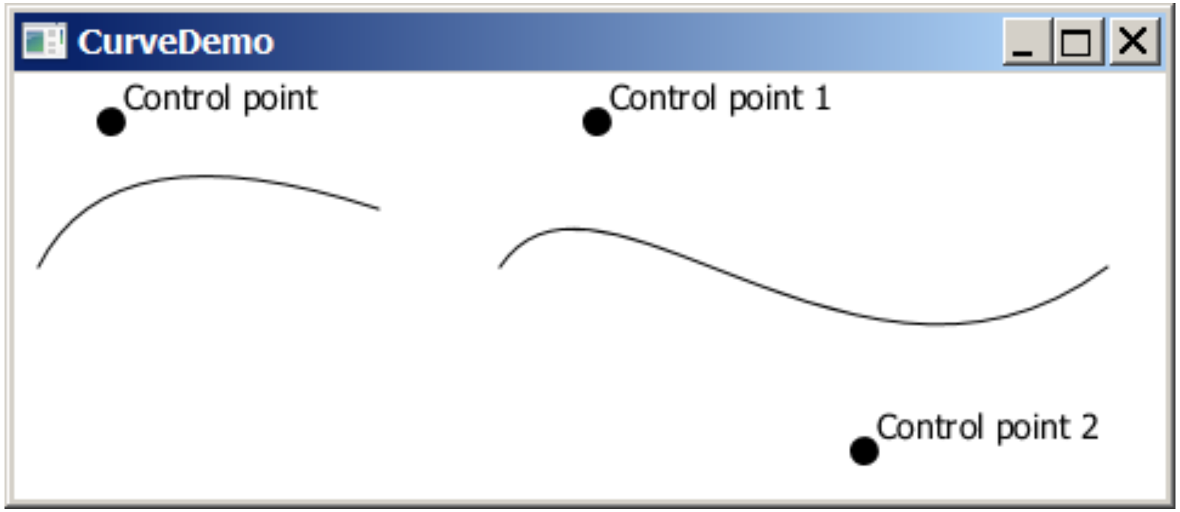


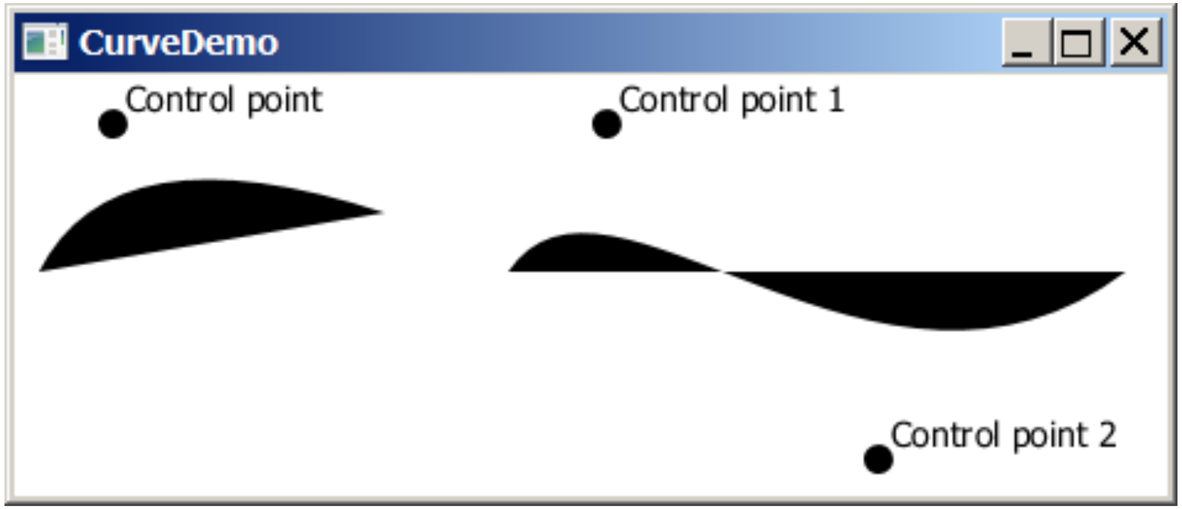
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.shape.CubicCurve	
-startX: DoubleProperty	The x-coordinate of the start point (default 0).
-startY: DoubleProperty	The y-coordinate of the start point (default 0)..
-endX: DoubleProperty	The x-coordinate of the end point (default 0)..
-endY: DoubleProperty	The y-coordinate of the end point (default 0)..
-controlX1: DoubleProperty	The x-coordinate of the first control point (default 0)..
-controlY1: DoubleProperty	The y-coordinate of the first control point (default 0)..
-controlX2: DoubleProperty	The x-coordinate of the second control point (default 0)..
-controlY2: DoubleProperty	The y-coordinate of the second control point (default 0)..
+CubicCurve()	Creates an empty cubic curve.
+CubicCurve(startX: double, startY: double, controlX1: double, controlY1: double, controlX2: double, controlY2: double, endX: double, endY: double)	Creates a cubic curve with the specified arguments.

CurveDemo

Curve Demo





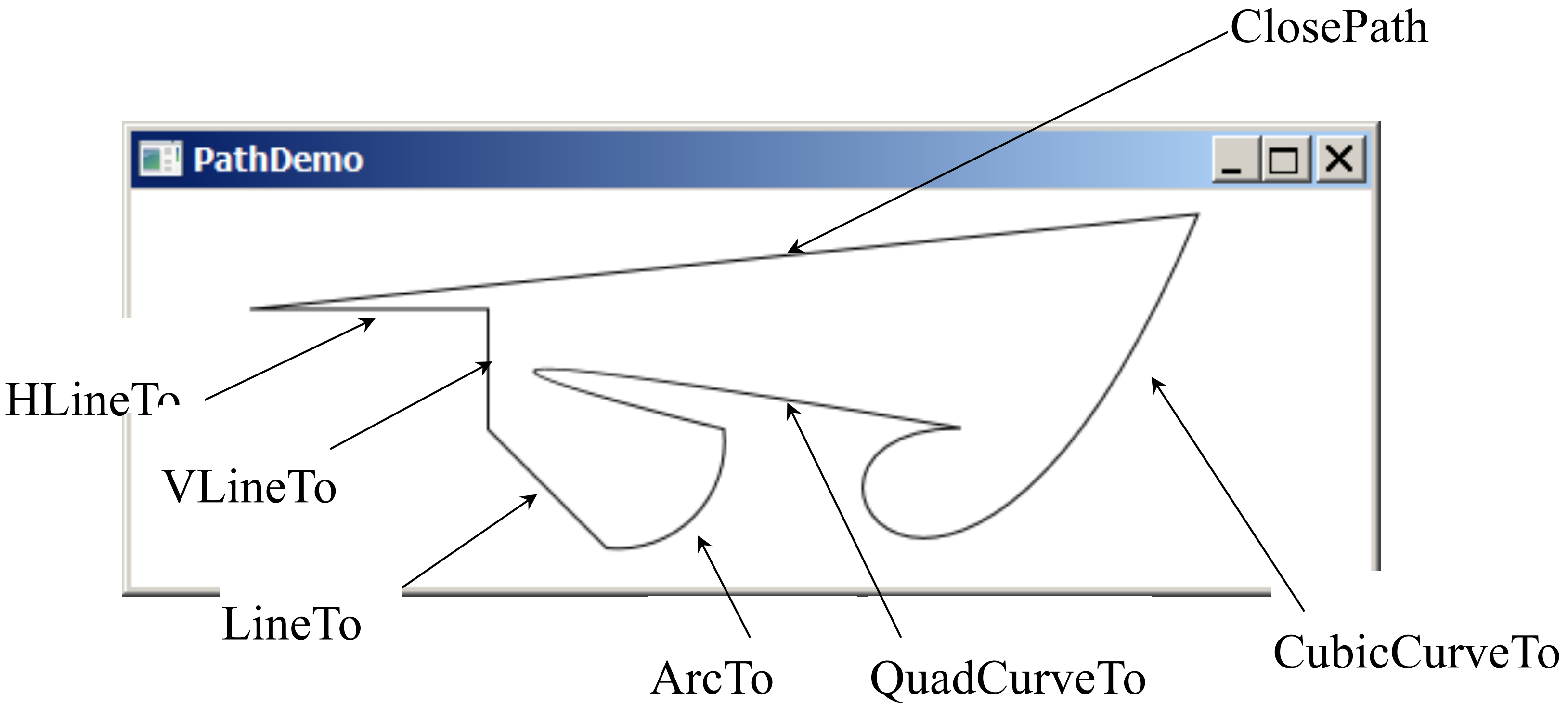
CurveDemo

Path

The **Path** class models an arbitrary geometric path. A path is constructed by adding path elements into the path. The **PathElement** is the root class for the path elements **MoveTo**, **HLineTo**, **VLineTo**, **LineTo**, **ArcTo**, **QuadCurveTo**, **CubicCurveTo**, and **ClosePath**.

PathDemo

Path



ClosePath

HLineTo

VLineTo

LineTo

ArcTo

QuadCurveTo

CubicCurveTo

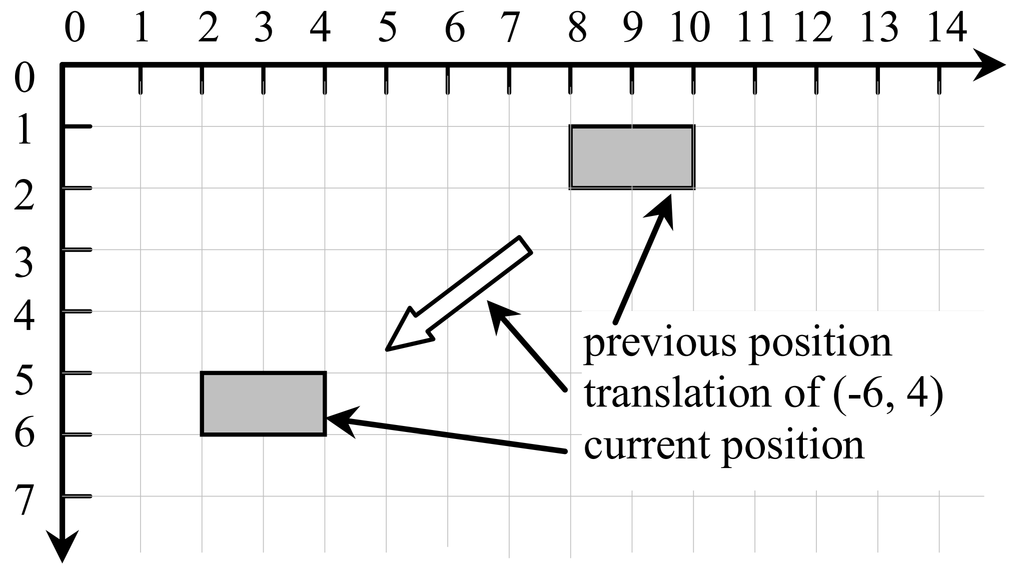
PathDemo

Coordinate Transformations

JavaFX supports coordinate transformations using translation, rotation, and scaling.

Translations

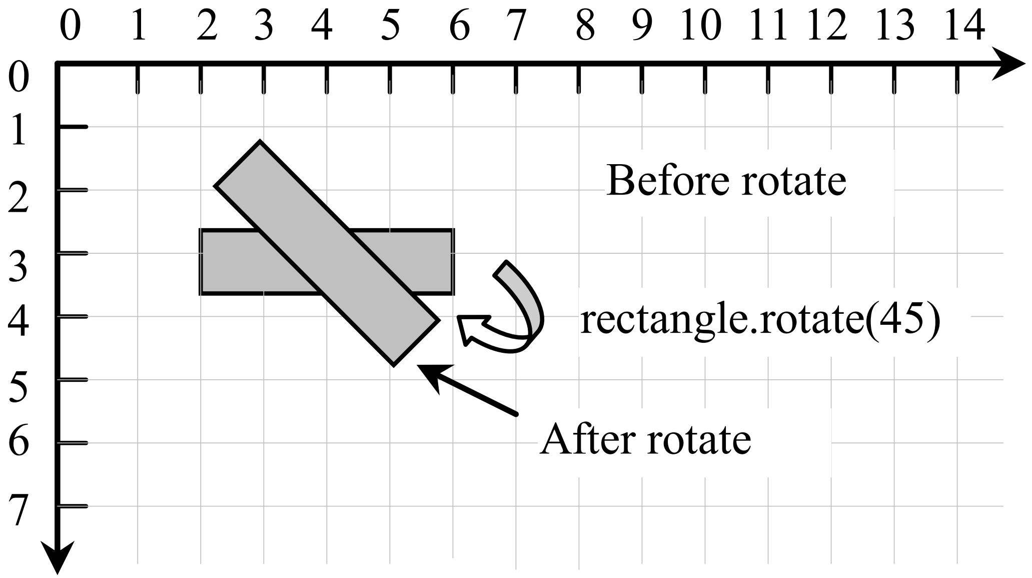
You can use the **setTranslateX(double x)**, **setTranslateY(double y)**, and **setTranslateZ(double z)**methods in the **Node** class to translate the coordinates for a node.



TranslationDemo

Rotations

You can use the **rotate(double theta)** method in the **Node** class to rotate a node by theta degrees from its pivot point clockwise, where theta is a double value in degrees. The pivot point is automatically computed based on the bounds of the node.



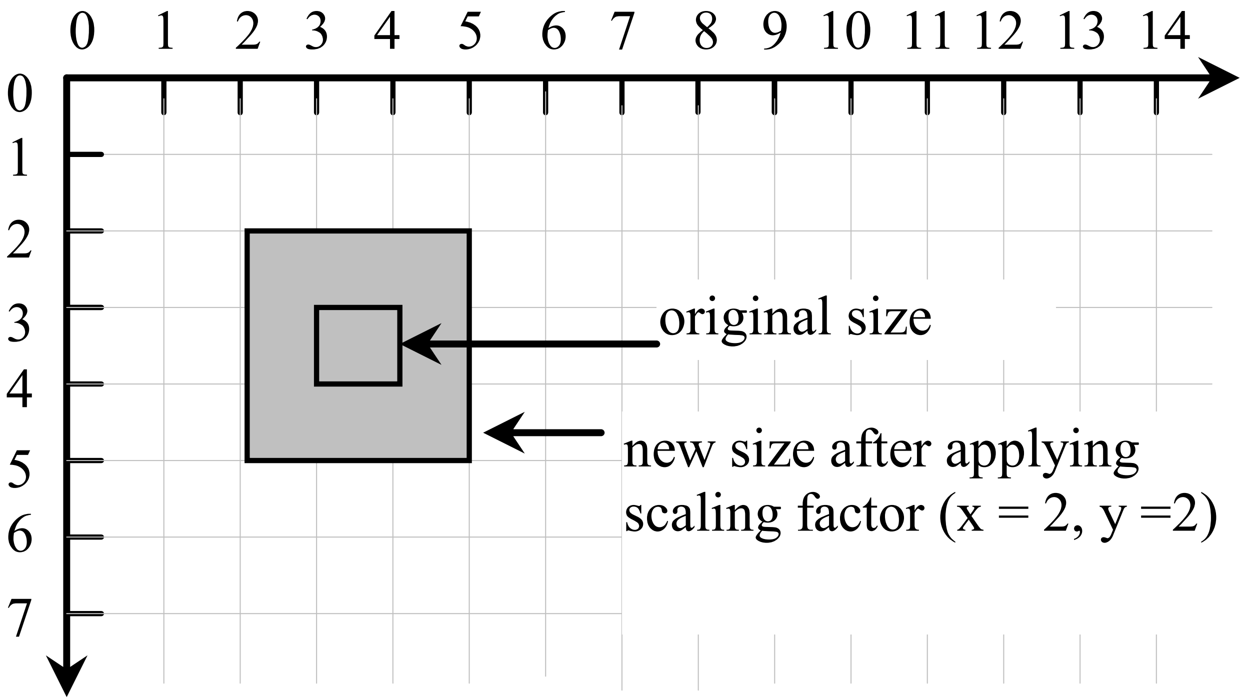
RotateDemo

Scaling

You can use the **setScaleX(double sx)**, **setScaleY(double sy)** , and **setScaleY(double sy)** methods in the **Node** class to specify a scaling factor. The node will appear larger or smaller depending on the scaling factor. Scaling alters the coordinate space of the node such that each unit of distance along the axis is multiplied by the scale factor. As with rotation transformations, scaling transformations are applied to enlarge or shrink the node around the pivot point.

For a node of the rectangle shape, the pivot point is the center of the rectangle.

Scaling



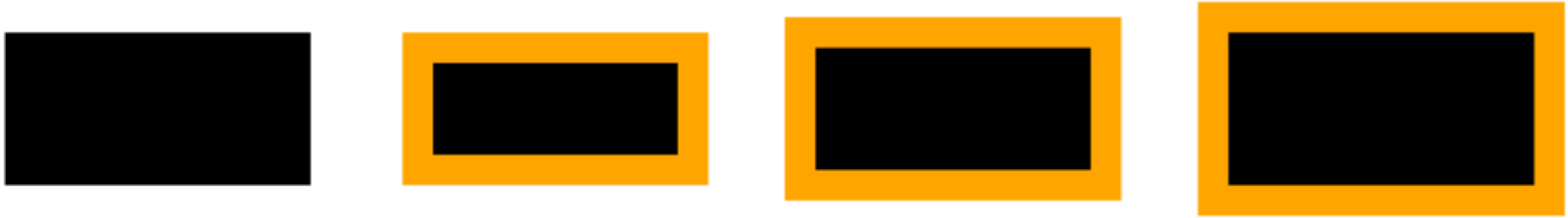
ScaleDemo

Strokes

javafx.scene.shape.Shape	
+setStroke(paint: Paint): void	Sets a paint for the stroke.
+setStrokeWidth(width: double): void	Sets a width for the stroke (default 1).
+setStrokeType(type: StrokeType): void	Sets a type for the stroke to indicate whether the stroke is placed inside, centered, or outside of the border (default: CENTERED).
+setStrokeLineCap(type: StrokeLineCap): void	Specifies the end cap style for the stroke (default: BUTT).
+setStrokeLineJoin(type: StrokeLineJoin): void	Specifies how two line segments are joined (default: MITER).
+getStrokeDashArray(): ObservableList<Double>	Returns a list that specifies a dashed pattern for line segments.
+setStrokeDashOffset(distance: double): void	Specifies the offset to the first segment in the dashed pattern.

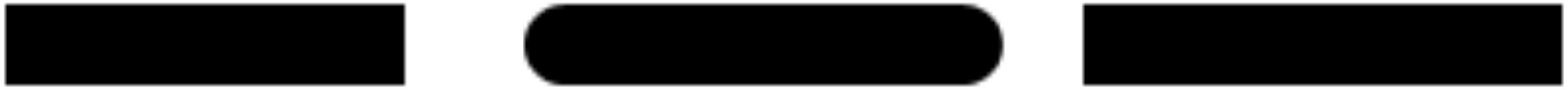
strokeType

The **setStrokeType(type)** method sets a type for the stroke. The type defines whether the stroke is inside, outside, or in the center of the border using the constants **StrokeType.INSIDE**, **StrokeType.CENTERED** (default), or **StrokeType.OUTSIDE**,



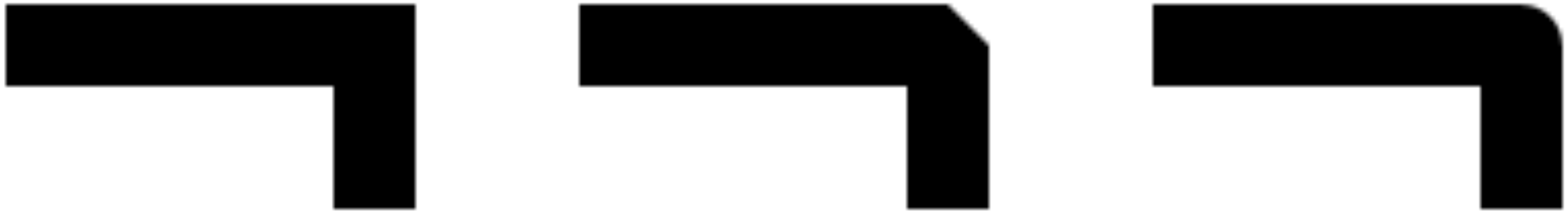
strokeLineCap

The **setStrokeType(type)** method sets a type for the stroke. The type defines whether the stroke is inside, outside, or in the center of the border using the constants **StrokeType.INSIDE**, **StrokeType.CENTERED** (default), or **StrokeType.OUTSIDE**,



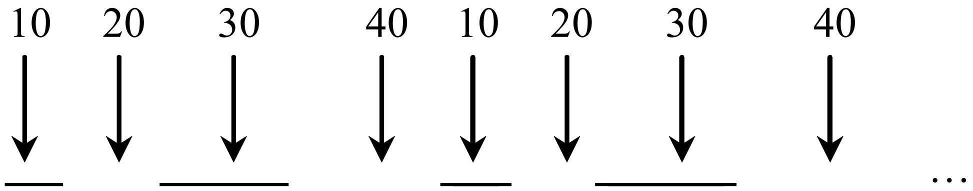
strokeLineJoin

The **setStrokeLineJoin** method defines the decoration applied where path segments meet. You can specify three types of line join using the constants **StrokeLineJoin.MITER** (default), **StrokeLineJoin.BEVEL**, and **StrokeLineJoin.ROUND**.



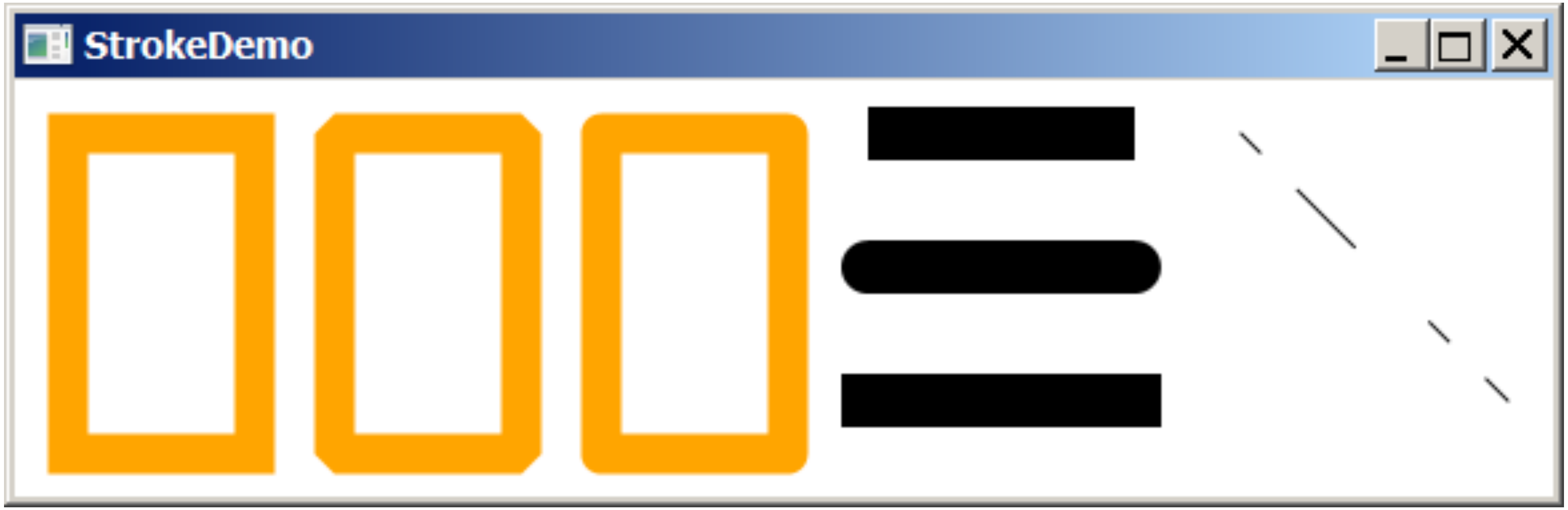
strokeDashArray

The **Shape** class has a property named **strokeDashArray** of the **ObservableList<Double>** type. This property is used to define a dashed pattern for the stroke. Alternate numbers in the list specify the lengths of the opaque and transparent segments of the dashes.



[10.0, 20.0, 30.0, 40.0]

Stroke Demo



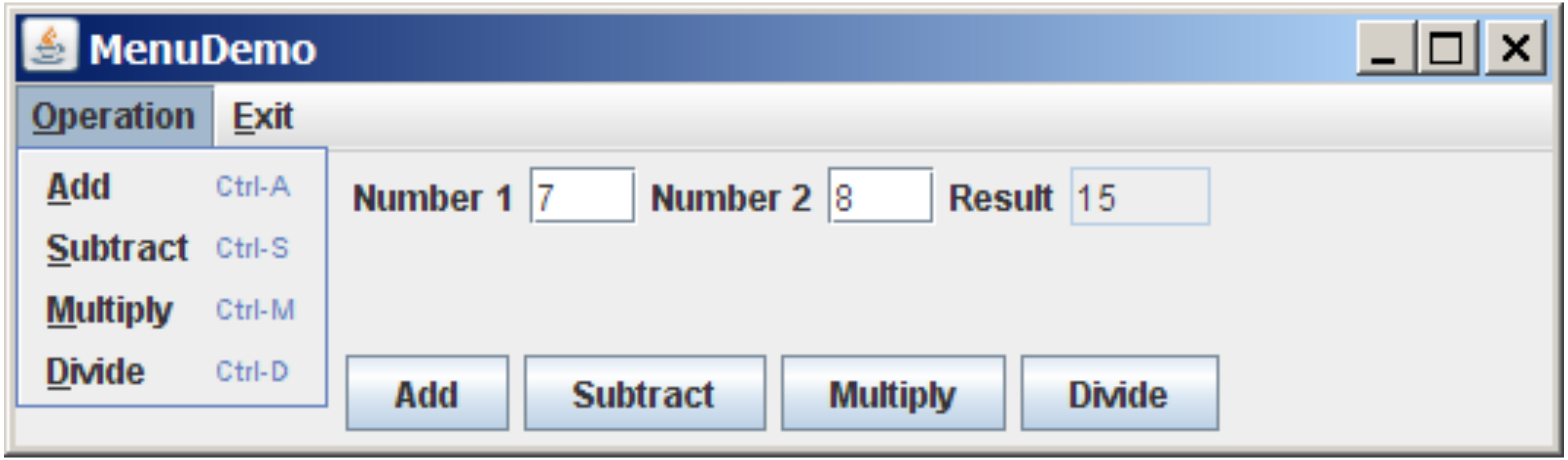
StrokeDemo

Menu

Menus make selection easier and are widely used in window applications. JavaFX provides five classes that implement menus: MenuBar, Menu, MenuItem, CheckMenuItem, and RadioButtonMenuItem.

MenuBar is a top-level menu component used to hold the menus. A menu consists of menu items that the user can select (or toggle on or off). A menu item can be an instance of MenuItem, CheckMenuItem, or RadioButtonMenuItem. Menu items can be associated with nodes and keyboard accelerators.

Creating Menus



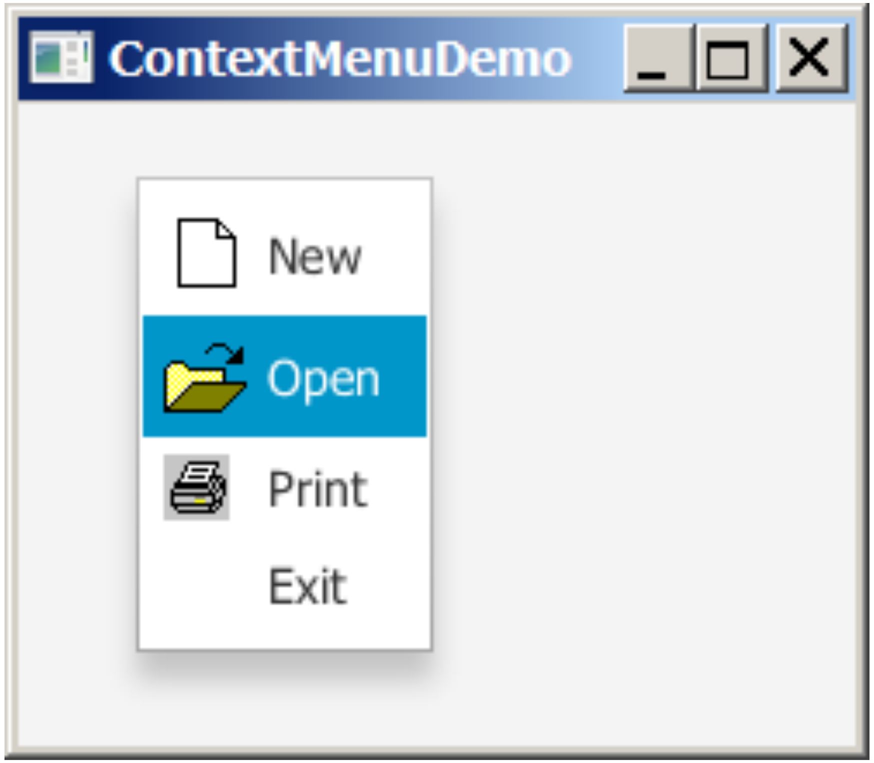
Number 1 7 Number 2 8 Result 15

MenuDemo

Context Menu

A *context menu*, also known as a *popup menu*, is like a regular menu, but does not have a menu bar and can float anywhere on the screen. Creating a context menu is similar to creating a regular menu. First, you create an instance of ContextMenu, then you can add MenuItem, CheckMenuItem, and RadioMenuItem to the context menu.

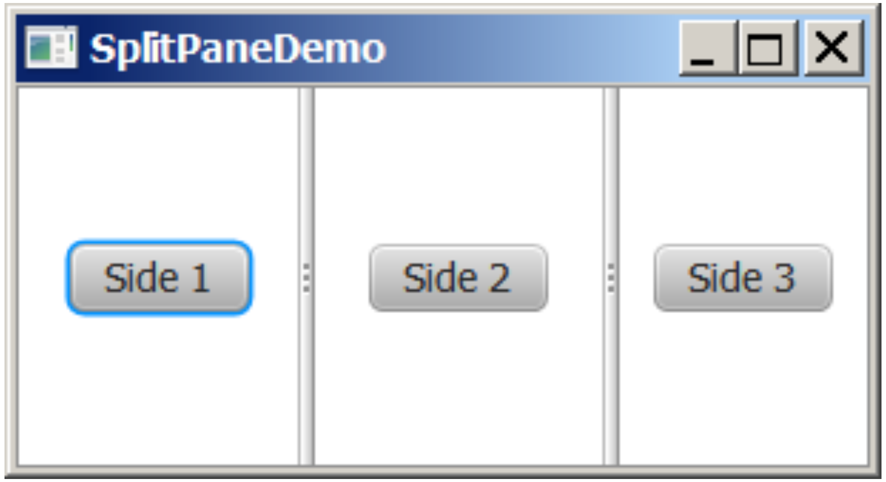
Creating Context Menus

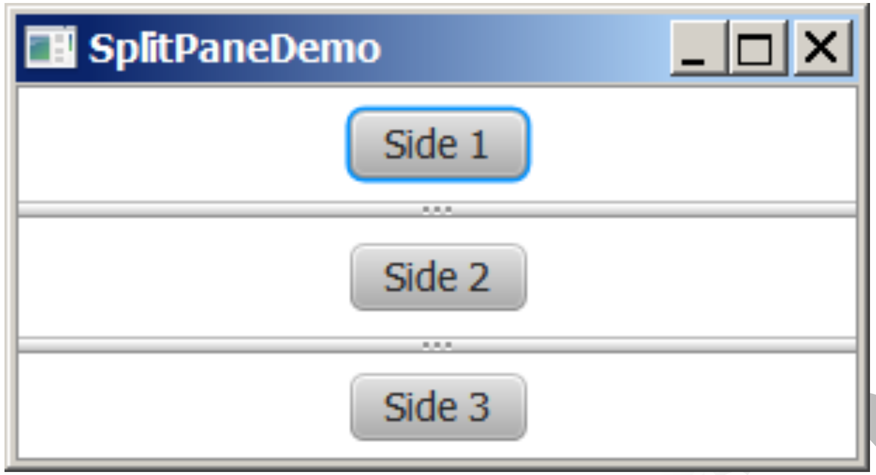


ContextMenuDemo

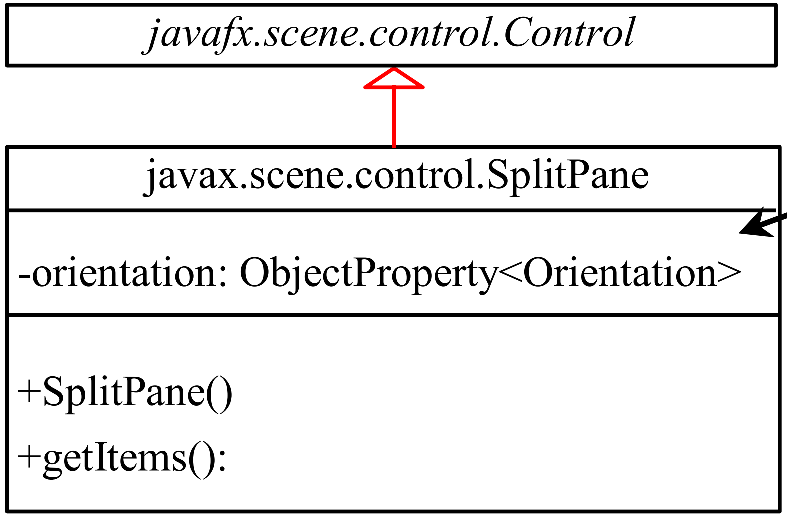
SplitPane

The **SplitPane** class can be used to display multiple panes and allow the user to adjust the size of the panes.





Using SplitPane

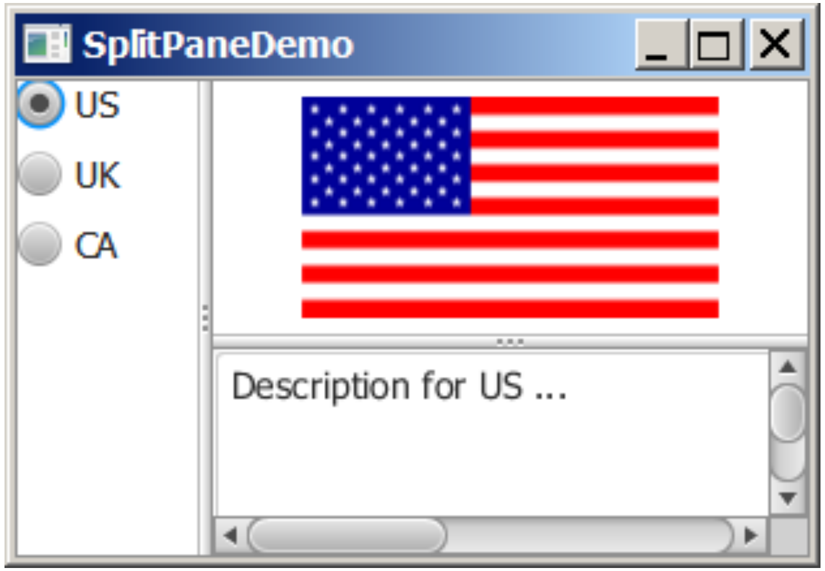

javafx.scene.control.Control
javax.scene.control.SplitPane
-orientation: ObjectProperty<Orientation>
+SplitPane()
+getItems():

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.

Specifies the orientation of the pane.

Constructs a default split pane with horizontal orientation.

Returns a list of items in the pane.

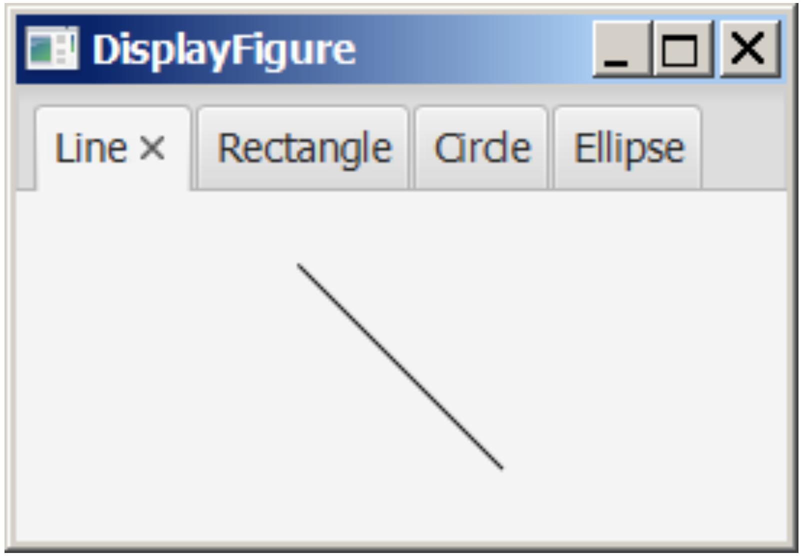


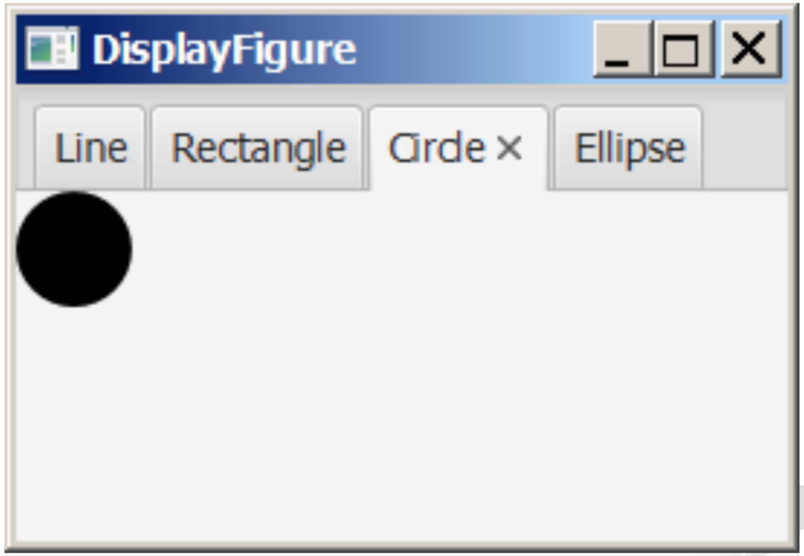
Description for US ...

SplitPaneDemo

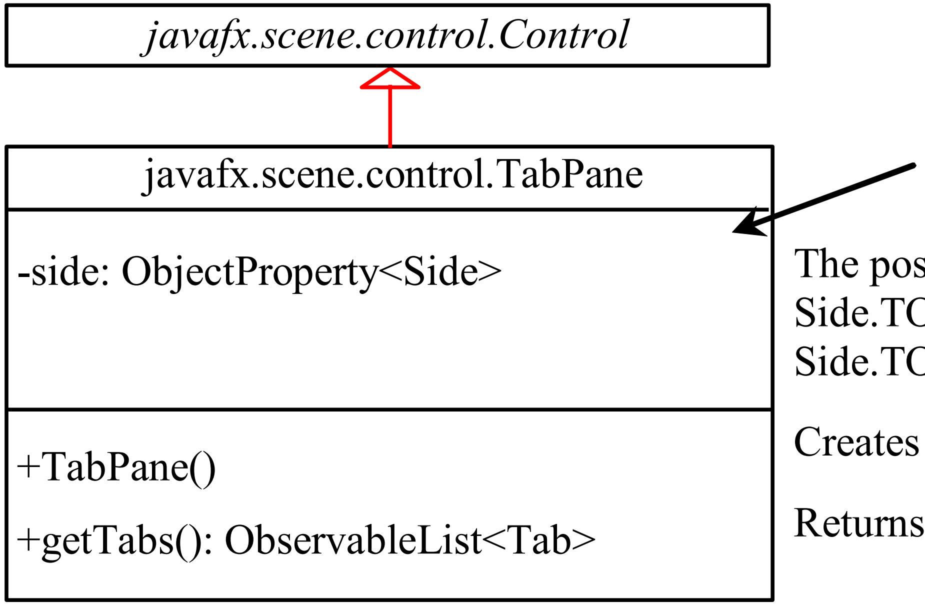
TabPane

The **TabPane** class can be used to display multiple panes with tabs.





The TabPane Class



javafx.scene.control.Control

javafx.scene.control.TabPane

-side: ObjectProperty<Side>

+TabPane()

+getTabs(): ObservableList<Tab>

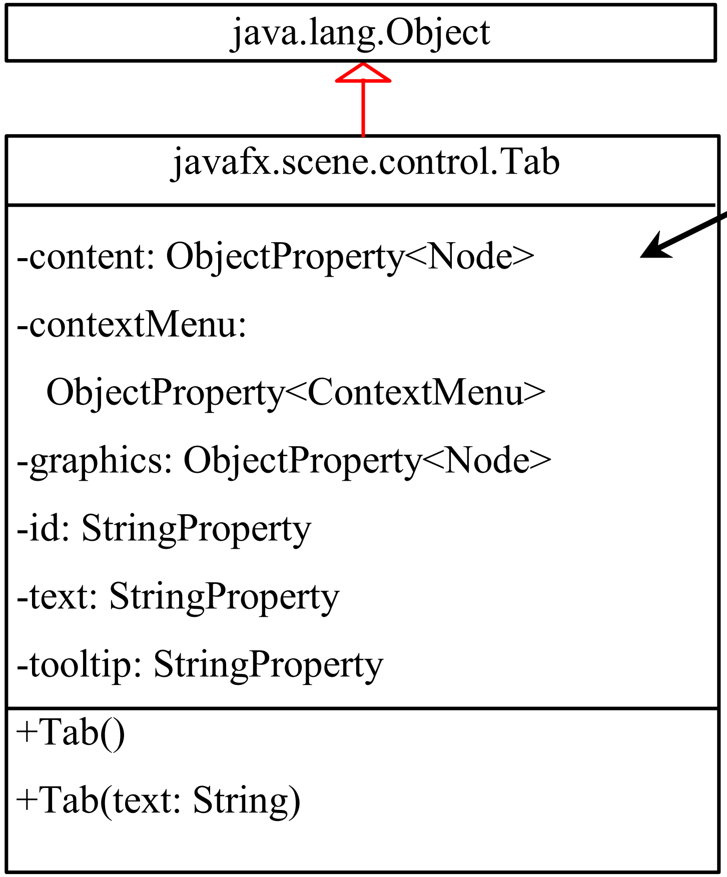
The `getter` and `setter` methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

The position of the tab in the tab pane. Possible values are: Side.TOP, Side.BOTTOM, Side.LEFT, and Side.RIGHT (default: Side.TOP).

Creates a default tab pane.

Returns a list of tabs in this tab pane.

The Tab Class



java.lang.Object

javafx.scene.control.Tab

-content: ObjectProperty<Node>

-contextMenu:
  ObjectProperty<ContextMenu>

-graphics: ObjectProperty<Node>

-id: StringProperty

-text: StringProperty

-tooltip: StringProperty

+Tab()

+Tab(text: String)

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.

The content associated with the tab.

The context menu associated with the tab.

The graphics in the tab.

The id for the tab.

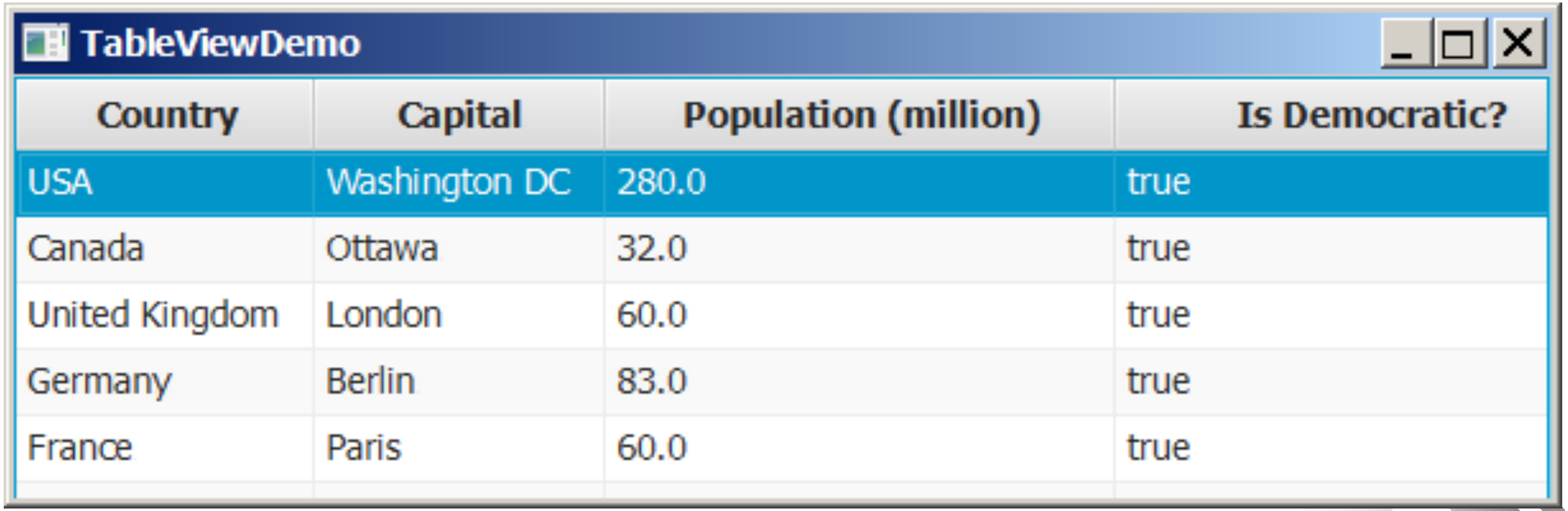
The text shown in the tab.

The tooltip associated with the tab.

TabPaneDemo

TableView

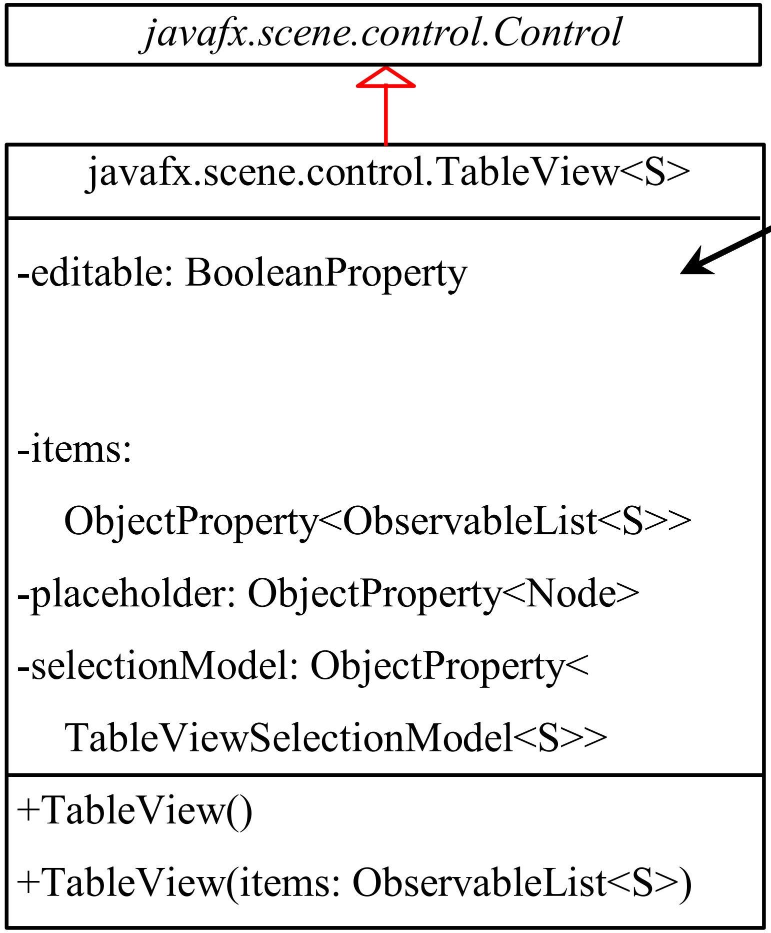
You can display tables using the **TableView** class.



Country	Capital	Population (million)	Is Democratic?
USA	Washington DC	280.0	true
Canada	Ottawa	32.0	true
United Kingdom	London	60.0	true
Germany	Berlin	83.0	true
France	Paris	60.0	true

TableViewDemo

The TableView Class



javafx.scene.control.Control

javafx.scene.control.TableView<S>

-editable: BooleanProperty

-items: ObjectProperty<ObservableList<S>>

-placeholder: ObjectProperty<Node>

-selectionModel: ObjectProperty< TableViewSelectionModel<S>>

+TableView()

+TableView(items: ObservableList<S>)

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.

Specifies whether this TableView is editable. For a cell to be editable, TableView, TableColumn, and TableCell for the cell should all be true.

The data model for the TableViee.

This Node is shown when table has no contents.

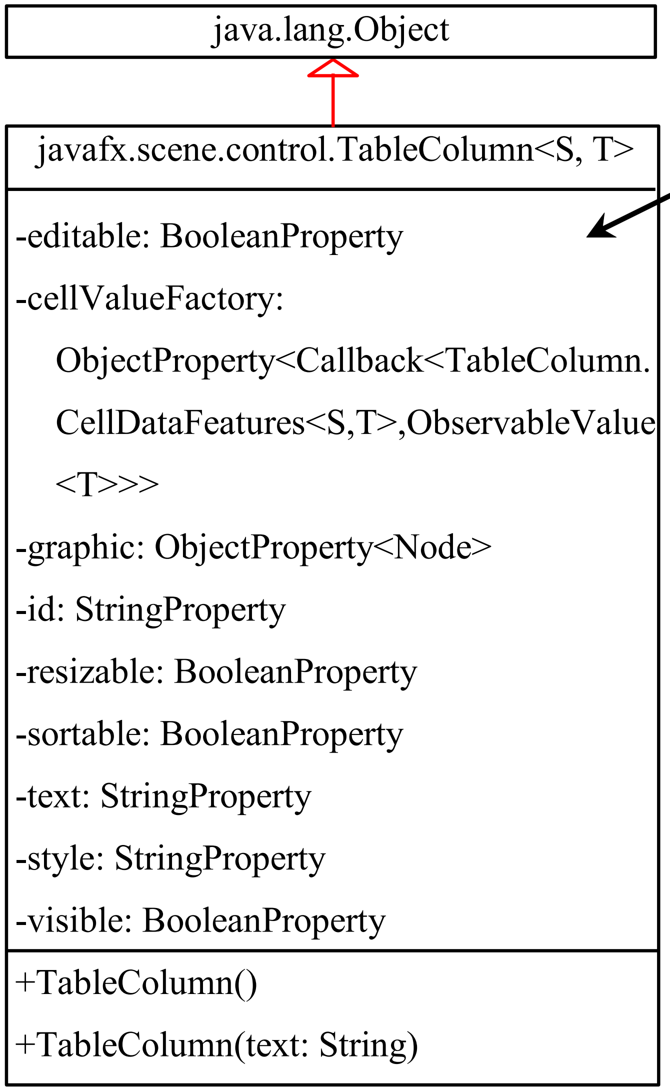
Specifies single or multiple selections.

Creates a default TableView with no content.

Creates a default TableView with the specified content.

The TableColumn 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.



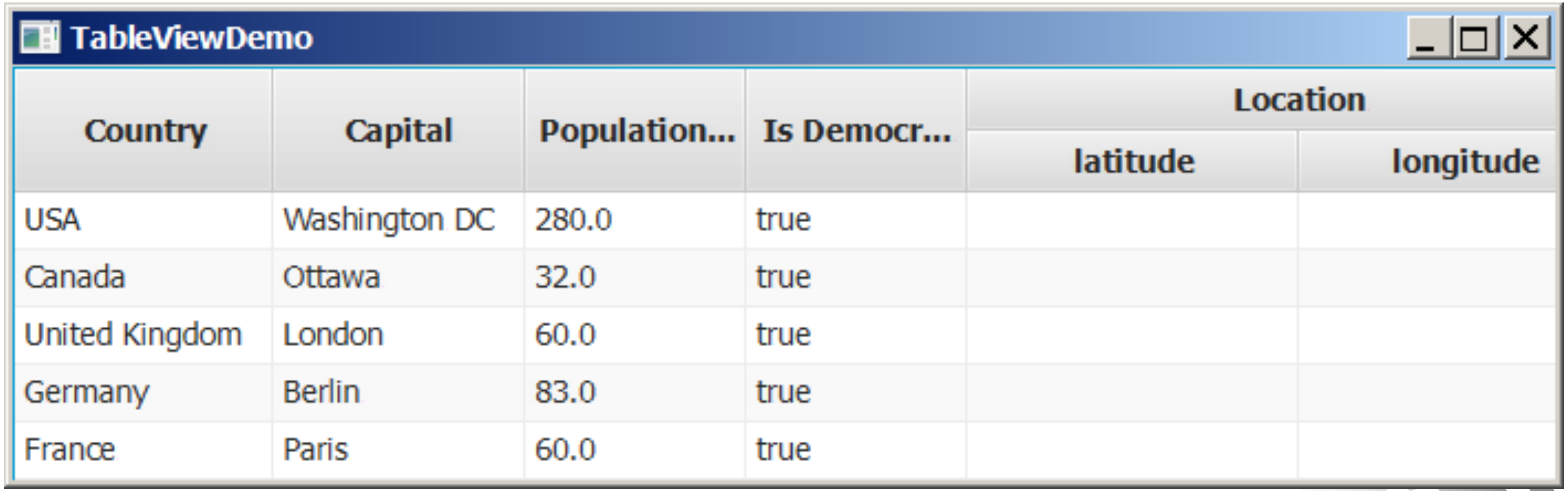
java.lang.Object

javafx.scene.control.TableColumn<S, T>

-editable: BooleanProperty	Specifies whether this TableColumn allows editing.
-cellValueFactory: ObjectProperty<Callback<TableColumn.CellDataFeatures<S,T>,ObservableValue<T>>>	The cell value factory to specify how to populate all cells within a single column.
-graphic: ObjectProperty<Node>	The graphic for this TableColumn.
-id: StringProperty	The id for this TableColumn.
-resizable: BooleanProperty	Indicates whether the column is resizable.
-sortable: BooleanProperty	Indicates whether the column is sortable.
-text: StringProperty	Text in the table column header.
-style: StringProperty	Specify the CSS style for the column.
-visible: BooleanProperty	Specify whether the column is visible (default: true).
+TableColumn()	Creates a default TableColumn.
+TableColumn(text: String)	Creates a TableView with the specified header text.

Add New Row

You can display tables using the **TableView** class.



Country	Capital	Population...	Is Democr...	Location	
				latitude	longitude
USA	Washington DC	280.0	true		
Canada	Ottawa	32.0	true		
United Kingdom	London	60.0	true		
Germany	Berlin	83.0	true		
France	Paris	60.0	true		

AddNewRowDemo