Chapter 34 Advanced JavaFX
Objectives

- To specify styles for UI nodes using JavaFX CSS (§34.2).
- To simplify creating JavaFX nodes using the builder classes (§34.3).
- To create quadratic curve, cubic curve, and path using the QuadCurve, CubicCurve, and Path classes (§34.4).
- To translation, rotation, and scaling to perform coordinate transformations for nodes (§34.5).
- To define a shape’s border using various types of strokes (§34.6).
- To create menus using the Menu, MenuItem, CheckMenuItem, and RadioMenuItem classes (§34.7).
- To create context menus using the ContextMenu class (§34.8).
- To use SplitPane to create adjustable horizontal and vertical panes (§34.9).
- To create tab panes using the TabPane control (§34.10).
- To create and display tables using the TableView and TableColumn classes (§34.11).
JavaFX CSS

A JavaFX style property is defined with a prefix –fx- to distinguish 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](http://docs.oracle.com/javafx/2/api/javafx.scene/doc-files/cssref.html). Listing 34.1 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. Multiple 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.
Builder Classes

The builder classes can be used to simplify creating JavaFX nodes. JavaFX provides a builder class for every node. Using the builder class can sometimes simplify coding. It is particularly useful when creating multiple objects of the same type with common properties.
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

```
javafx.scene.shape.QuadCurve

-startX: DoubleProperty
-startY: DoubleProperty
-endX: DoubleProperty
-endY: DoubleProperty
-controlX: DoubleProperty
-controlY: DoubleProperty

+QuadCurve()
+QuadCurve(startX: double,
    startY: double, controlX: double, controlY: double,
    endX: double, endY: double)
```

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 x-coordinate of the start point (default 0).
The y-coordinate of the start point (default 0).
The x-coordinate of the end point (default 0).
The y-coordinate of the end point (default 0).
The x-coordinate of the control point (default 0).
The y-coordinate of the control point (default 0).

Creates an empty quad curve.

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:

The `CubicCurve` class represents a cubic curve in JavaFX. It is used to draw smooth curves in a graphical user interface. The class provides methods to set and get the properties of the curve, such as the start point, end point, and control points.

```java
javafx.scene.shape.CubicCurve

-startX: DoubleProperty
-startY: DoubleProperty
-endX: DoubleProperty
-endY: DoubleProperty
-controlX1: DoubleProperty
-controlY1: DoubleProperty
-controlX2: DoubleProperty
-controlY2: DoubleProperty

+CubicCurve()
+CubicCurve(startX: double, startY: double, controlX1: double, controlY1: double, controlX2: double, controlY2: double, endX: double, endY: double)
```

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 `startX` and `startY` properties represent the x and y coordinates of the start point of the curve (default 0).
- The `endX` and `endY` properties represent the x and y coordinates of the end point of the curve (default 0).
- The `controlX1` and `controlY1` properties represent the x and y coordinates of the first control point (default 0).
- The `controlX2` and `controlY2` properties represent the x and y coordinates of the second control point (default 0).

The `CubicCurve()` constructor creates an empty cubic curve.

The `CubicCurve(startX: double, startY: double, controlX1: double, controlY1: double, controlX2: double, controlY2: double, endX: double, endY: double)` constructor creates a cubic curve with the specified arguments.

The `CurveDemo` program demonstrates the usage of the `CubicCurve` class.
Curve Demo
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.
Path

- HLineTo
- VLineTo
- LineTo
- ArcTo
- QuadCurveTo
- CubicCurveTo
- ClosePath

PathDemo Run
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.

![Graph showing translation example](image)
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.

```
Run RotationDemo
rectangle.rotate(45)
```
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

Run ScaleDemo

original size

new size after applying scaling factor (x = 2, y = 2)
# Strokes

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

![MenuDemo app](image)
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](image)

Run `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

```
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.

Run SplitPaneDemo
TabPane

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

```java
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.
TableView

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

![TableViewDemo](image)

<table>
<thead>
<tr>
<th>Country</th>
<th>Capital</th>
<th>Population (million)</th>
<th>Is Democratic?</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA</td>
<td>Washington DC</td>
<td>280.0</td>
<td>true</td>
</tr>
<tr>
<td>Canada</td>
<td>Ottawa</td>
<td>32.0</td>
<td>true</td>
</tr>
<tr>
<td>United Kingdom</td>
<td>London</td>
<td>60.0</td>
<td>true</td>
</tr>
<tr>
<td>Germany</td>
<td>Berlin</td>
<td>83.0</td>
<td>true</td>
</tr>
<tr>
<td>France</td>
<td>Paris</td>
<td>60.0</td>
<td>true</td>
</tr>
</tbody>
</table>

**Run**
The TableView Class

```java
javafx.scene.control.Control
```

```java
javafx.scene.control.TableView<S>
```

- `editable`: `BooleanProperty`
- `items`: `ObjectProperty<ObservableList<S>>`
- `placeholder`: `ObjectProperty<Node>`
- `selectionModel`: `ObjectProperty<TableViewSelectionModel<S>>`

+ `TableView()`
+ `TableView(items: ObservableList<S>)`

The `get` and `set` 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

java.lang.Object

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

- editable: BooleanProperty
- cellValueFactory: ObjectProperty<Callback<TableColumn.CellDataFeatures<S, T>, ObservableValue<T>>>
- graphic: ObjectProperty<Node>
- id: StringProperty
- resizable: BooleanProperty
- sortable: BooleanProperty
- text: StringProperty
- style: StringProperty
- visible: BooleanProperty

+ TableColumn()
+ TableColumn(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.

Specifies whether this TableColumn allows editing.
The cell value factory to specify how to populate all cells within a single column.

The graphic for this TableColumn.
The id for this TableColumn.
Indicates whether the column is resizable.
Indicates whether the column is sortable.
Text in the table column header.
Specify the CSS style for the column.
Specify whether the column is visible (default: true).
Creates a default TableColumn.
Creates a TableView with the specified header text.
Add New Row

You can display tables using the `TableView` class.