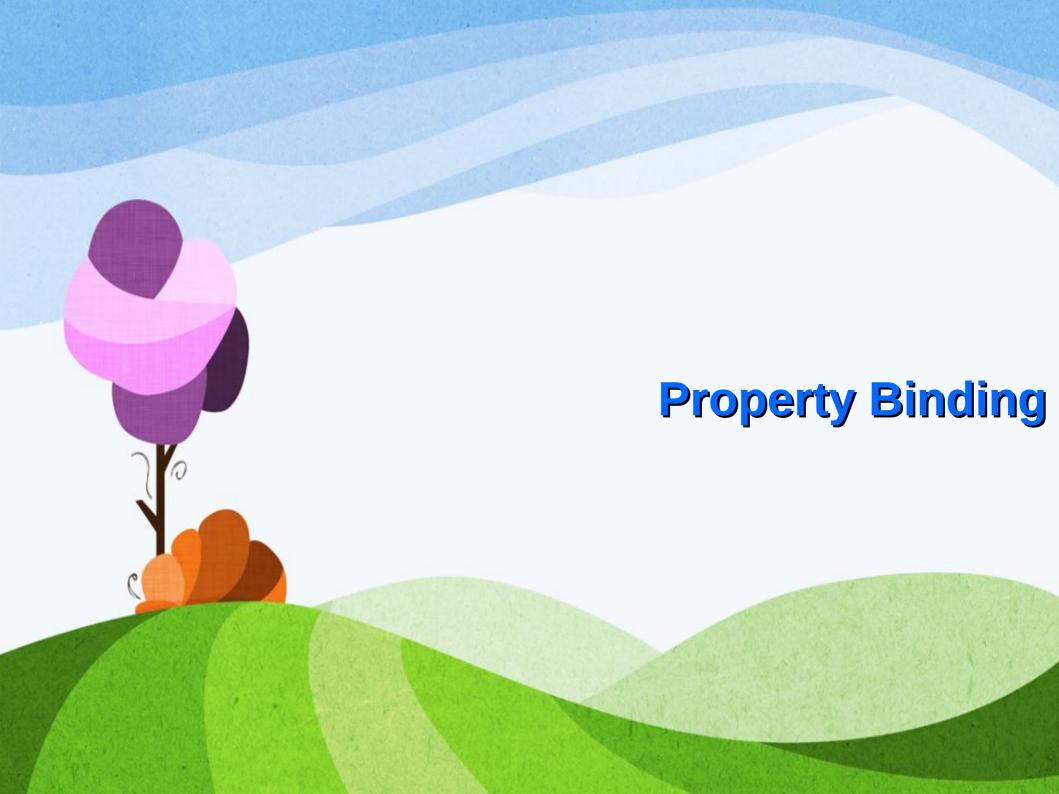


California State University, Los Angeles Computer Science Department

Lecture XI
JavaFX III



Property Binding

- property binding: enables a target object to be bound to a source object
 - target object is called the binding object or binding property
 - source object is called a bindable object or observable object

A change to the source object will be automatically reflected in the target object.

ShowCircleCentered.java

LISTING 14.5 ShowCircleCentered.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 ShowCircleCentered extends Application {
 9
      @Override // Override the start method in the Application class
10
      public void start(Stage primaryStage) {
        // Create a pane to hold the circle
11
12
        Pane pane = new Pane();
13
14
        // Create a circle and set its properties
15
        Circle circle = new Circle():
16
        circle.centerXProperty().bind(pane.widthProperty().divide(2));
17
        circle.centerYProperty().bind(pane.heightProperty().divide(2));
18
        circle.setRadius(50);
19
        circle.setStroke(Color.BLACK);
20
        circle.setFill(Color.WHITE):
21
        pane.getChildren().add(circle); // Add circle to the pane
22
23
        // Create a scene and place it in the stage
24
        Scene scene = new Scene(pane, 200, 200);
25
        primaryStage.setTitle("ShowCircleCentered"); // Set the stage title
26
        primaryStage.setScene(scene); // Place the scene in the stage
27
        primaryStage.show(); // Display the stage
28
29
```

ShowCircleCentered.java

- same as the previous example, except binds circle's centerX and centerY properties to half of pane's width and height.
- circle.centerXProperty() returns centerX
- pane.widthPropery() returns width
- both centerX and width are binding properties of the DoubleProperty type
- all of the number binding property classes contain add, subtract, multiply, and divide methods for basic math operations and return a new observable property
- pane.widthProperty().divide(2) returns a new observable property that represents half of the pane's width
- since centerX is bound to width.divide(2), when pane's width is changed, centerX automatically updates itself to match pane's width / 2

Property Binding

- © Circle class has a centerX property for representing the x-coordinate of the circle.
 - can be used as both target and source in property binding (as can many other JavaFX class properties)

A target "listens" for changes to the source and automatically updates itself once a chance is made in the source.

The bind() Method

To bind a source to a target use the bind method: target.bind(source);

- bind() is defined in javafx.beans.property.Property interface.
 - a binding property is an instance of Property

- a source object is an instance of javafx.beans.value.ObservableValue
 - an ObservableValue is an entity that wraps a value and allows to observe the value for changes.

Primitive Types and Strings

JavaFX defines binding properties for primitive types and strings

- DoubleProperty, FloatProperty, LongProperty, IntegerProperty, BooleanProperty, StringProperty
 - these are all subtypes of ObservableValue so they can also be used as source objects for binding

Getters and Setters for Binding Properties

- By convention, each binding property (i.e. centerX) in a JavaFX class has a getter (getCenterX()) and a setter (setCenterX(double)).
- There is also a getter for the property itself.
 - The naming convention for this method is the property name followed by the word **Property**
 - Example: the property getter method for centerX is centerXProperty()
- getCenterX() is a value getter method
 - returns a **double** value
- setCenterX() is a value setter method
- centerXProperty() is a property getter method
 - returns an object of the DoubleProperty type

Getters and Setters for Property Binding

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

(a) **x** is a binding property

(b) centerX is binding property

FIGURE 14.7 A binding property has a value getter method, setter method, and property getter method.

Binding Demo. java

LISTING 14.6 BindingDemo.java

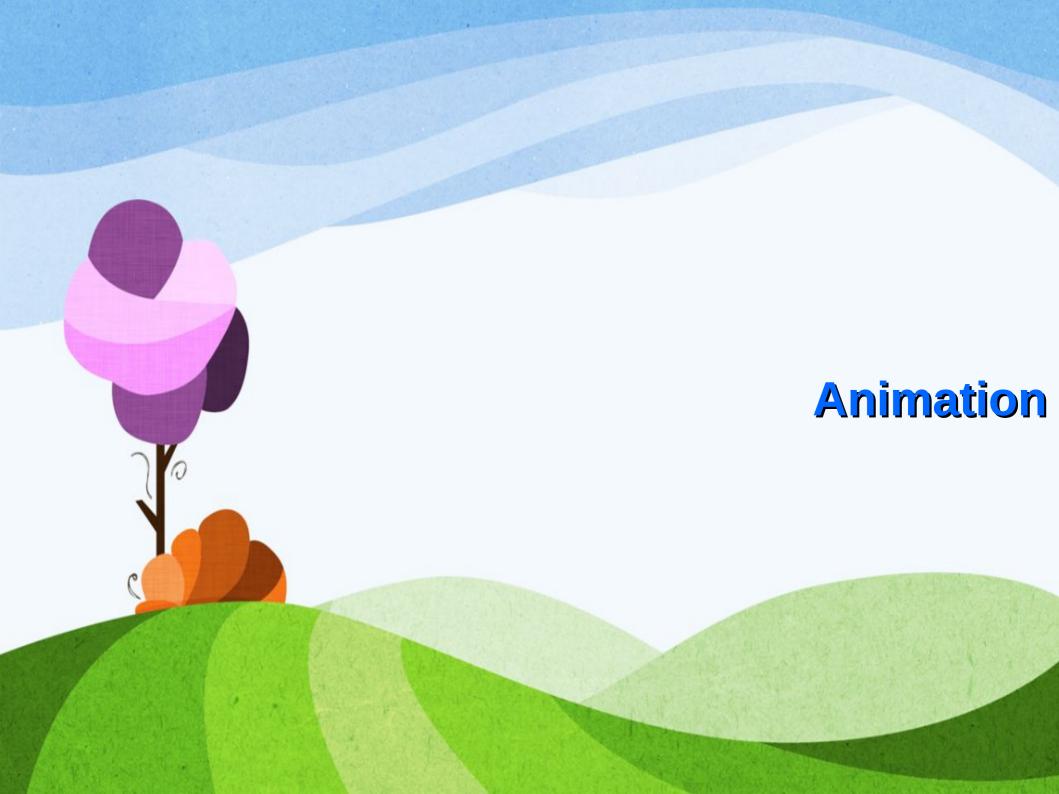
```
import javafx.beans.property.DoubleProperty;
    import javafx.beans.property.SimpleDoubleProperty;
 2 3 4 5 6 7
    public class BindingDemo {
      public static void main(String[] args) {
        DoubleProperty d1 = new SimpleDoubleProperty(1);
        DoubleProperty d2 = new SimpleDoubleProperty(2);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

Binding Demo. java

- line 6: creates an instance of DoubleProperty
 - uses SimpleDoubleProperty because numeric property classes are abstract.
 - Simple<Type>Property subclasses are concrete subclasses (substitute <Type> with a type i.e. Double, Integer, Boolean, etc.)
- line 8: binds d1 with d2 so values of d1 and d2 are the same
 - any changes to d2 will also update d1
- line 11: changes the value of d2

Unidirectional and Bidirectional Binding

- unidirectional binding: binding in only one direction, only changes in the source property will change the target property, changes to target will NOT change the source
 - example: changes to d2 will change d1, changes to d1 will not change d2
- bidirectional binding: binding in two directions, changes to one will affect the other and vice versa
 - example: changes to d2 will change d1, changes to d1 will change d2
 - only valid if both properties are both binding properties and observable properties, then you can bind them with the bindBidirectional method



Animation

JavaFX has an Animation class with functionality for all animations.

- © Generally if you want to animate something you should use a subclass of the **Animation** class.
 - PathTransition
 - FadeTransition
 - Timeline

Animation

See Code: FlagRisingAnimation.java

javafx.animation.Animation

-autoReverse: BooleanProperty

-cycleCount: IntegerProperty

-rate: DoubleProperty

-status: ReadOnlyObjectProperty

<Animation.Status>

+pause(): void

+play(): void

+stop(): void

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.

Defines whether the animation reverses direction on alternating cycles.

Defines the number of cycles in this animation.

Defines the speed and direction for this animation.

Read-only property to indicate the status of the animation.

Pauses the animation.

Plays the animation from the current position.

Stops the animation and resets the animation.

FIGURE 15.15 The abstract Animation class is the root class for JavaFX animations.

Animation

- autoReverse is a Boolean property that indicates whether an animation will reverse its direction on the next cycle.
- cycleCount indicates the number of the cycles for the animation.
 - use the constant **Timeline.INDEFINTE** to indicate an indefinite number of cycles.
- rate defines the speed and direction of the animation.
 - negative and positive rates go in opposite directions
- status is a read-only property that indicates
 - Animation.Status.PAUSED
 - Animation.Status.RUNNING,
 - Animation.Status.STOPPED).
- The methods pause(), play(), and stop() do what you think they do.

PathTransition

animates the movement of a node along a path from one end of the path to the other over a given time.

javafx.animation.PathTransition

-duration: ObjectProperty<Duration>

-node: ObjectProperty<Node>

-orientation: ObjectProperty
 <PathTransition.OrientationType>

-path: ObjectType<Shape>

+PathTransition()

+PathTransition(duration: Duration, path: Shape)

+PathTransition(duration: Duration,

path: Shape, node: Node)

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 duration of this transition.

The target node of this transition.

The orientation of the node along the path.

The shape whose outline is used as a path to animate the node move.

Creates an empty PathTransition.

Creates a PathTransition with the specified duration and path.

Creates a PathTransition with the specified duration, path, and node.

FIGURE 15.16 The **PathTransition** class defines an animation for a node along a path.

PathTransition

Duration is an immutable class to define a duration of time. Has the following constants:

INDEFINTE an indefinite duration

ONE 1 million seconds

UNKNOWN unknown duration

ZERO 0 duration

add(), subtract(), multiply(), and divide()
methods to perform arithmetic..

* toHours(), toMinutes(), toSeconds(), and toMillis() return the number of hours, minutes, seconds, and milliseconds in this duration.

PathTransition

- PathTransition defines two constants:
 - NONE
 - ORTHOGONAL_TO_TANGENT specifies that the node is kept perpendicular to the path's tangent along the geometric path.

See Code:

- PathTransitionDemo.java
- FlagRisingAnimation.java

FadeTransition

Animates the change of the opacity of a node over a given time.

javafx.animation.FadeTransition

-duration: ObjectProperty<Duration>

-node: ObjectProperty<Node>

-fromValue: DoubleProperty

-toValue: DoubleProperty

-byValue: DoubleProperty

+FadeTransition()

+FadeTransition(duration: Duration)

+FadeTransition(duration: Duration,

node: Node)

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 duration of this transition.

The target node of this transition.

The start opacity for this animation.

The stop opacity for this animation.

The incremental value on the opacity for this animation.

Creates an empty FadeTransition.

Creates a FadeTransition with the specified duration.

Creates a FadeTransition with the specified duration and node.

FIGURE 15.18 The **FadeTransition** class defines an animation for the change of opacity in a node.

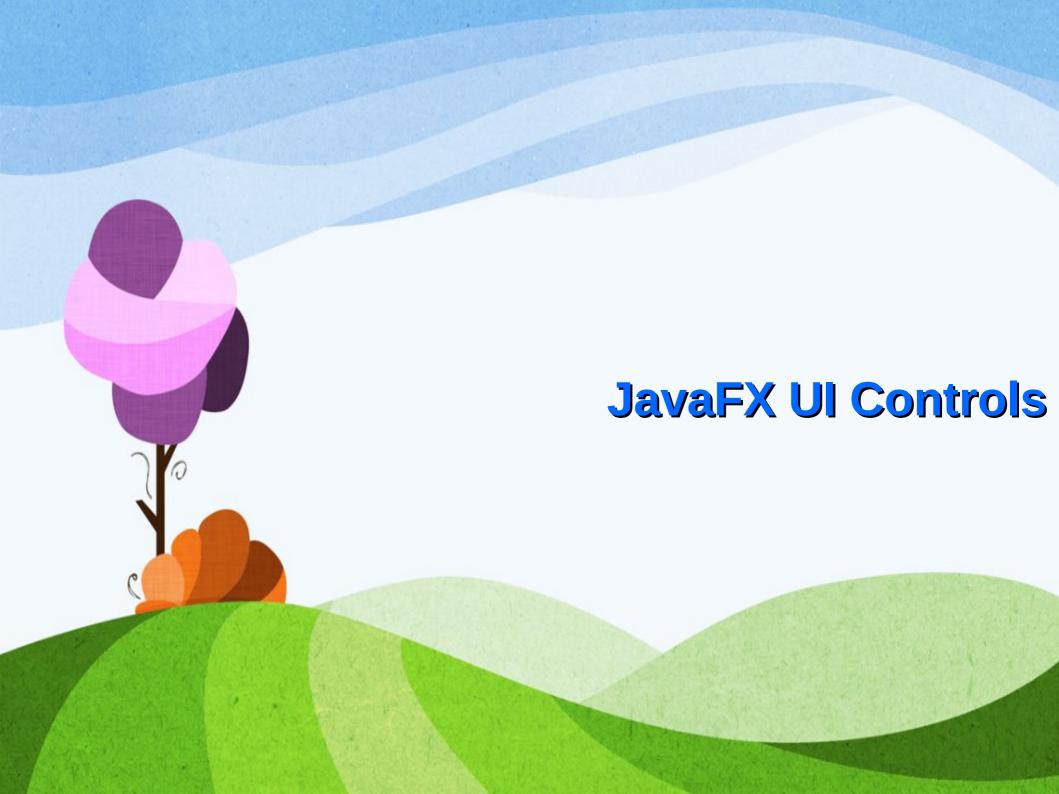
See Code: FadeTransitionDemo.java

TimeLine

Used to program any animation using one or more KeyFrames.

Each KeyFrame is executed sequentially at a given interval of time.

- The constructor for a KeyFrame takes an EventHandler called onFinished
 - this is called when the duration for the key frame has elapsed.
- See Code:
 - TimelineDemo.java
 - ClockAnimation.java
 - BouncingBallControl.java



Common JavaFX UI Controls

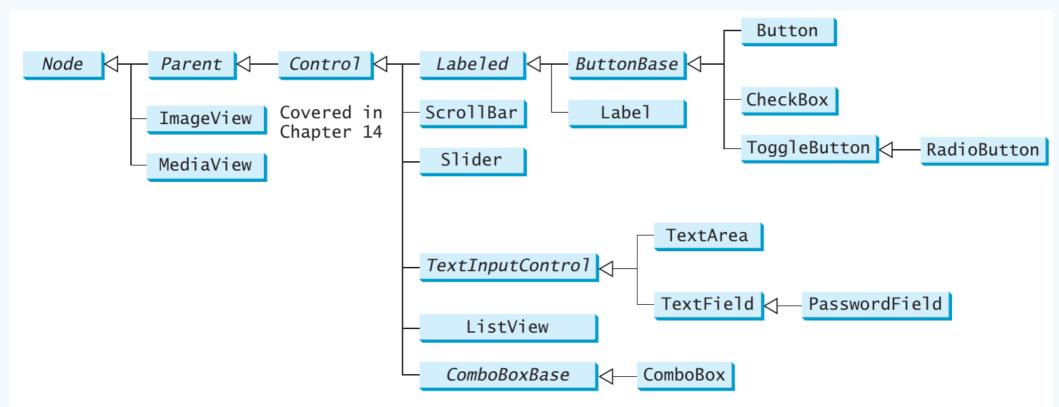


FIGURE 16.1 These UI controls are frequently used to create user interfaces.

Labeled and Label

- a label is used to display short text, a node, or both
- usually used to label a textfield.
- The Labeled class defines many common properties.

javafx.scene.control.Labeled

-alignment: ObjectProperty<Pos>

-contentDisplay:

ObjectProperty<ContentDisplay>

-graphic: ObjectProperty<Node>

-graphicTextGap: DoubleProperty

-textFill: ObjectProperty<Paint>

-text: StringProperty

-underline: BooleanProperty

-wrapText: BooleanProperty

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 alignment of the text and node in the labeled.

Specifies the position of the node relative to the text using the constants TOP, BOTTOM, LEFT, and RIGHT defined in ContentDisplay.

A graphic for the labeled.

The gap between the graphic and the text.

The paint used to fill the text.

A text for the labeled.

Whether text should be underlined.

Whether text should be wrapped if the text exceeds the width.

Labeled and Label

javafx.scene.control.Labeled

javafx.scene.control.Label

+Label()

+Label(text: String)

+Label(text: String, graphic: Node)

Creates an empty label.

Creates a label with the specified text.

Creates a label with the specified text and graphic.

Figure 16.3 Label is created to display a text or a node, or both.

See Code: LabelWithGraphic.java

Button

javafx.animation.Animation

-autoReverse: BooleanProperty

-cycleCount: IntegerProperty

-rate: DoubleProperty

-status: ReadOnlyObjectProperty

<Animation.Status>

+pause(): void

+play(): void

+stop(): void

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.

Defines whether the animation reverses direction on alternating cycles.

Defines the number of cycles in this animation.

Defines the speed and direction for this animation.

Read-only property to indicate the status of the animation.

Pauses the animation.

Plays the animation from the current position.

Stops the animation and resets the animation.

FIGURE 15.15 The abstract Animation class is the root class for JavaFX animations.

See Code: ButtonDemo.java

CheckBox

- inherits from ButtonBase and Labeled
- when a check box is clicked (checked or unchecked) it fires an ActionEvent.
- use isSelected() to see if the checkbox is selected.
- See Code: CheckBoxDemo.java

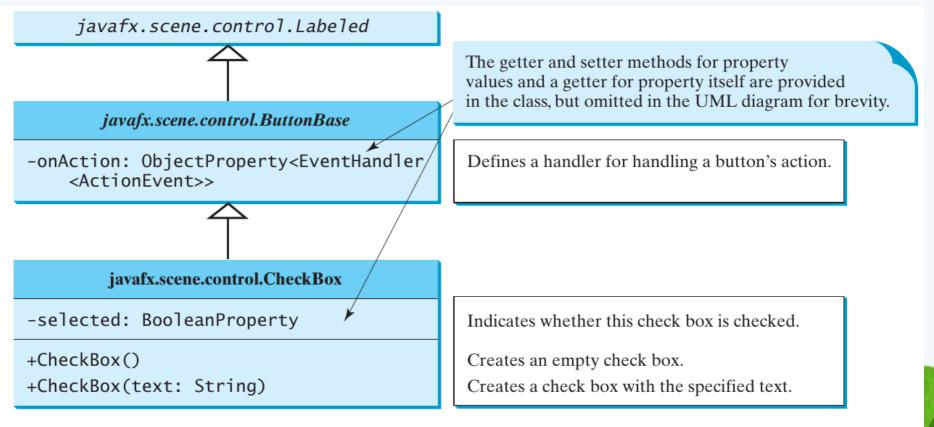


FIGURE 16.7 CheckBox contains the properties inherited from **ButtonBase** and **Labeled**.

RadioButton

Subclass of ToggleButton

- RadioButton is displayed as a circle
- ToggleButton is displayed similar to a button.

javafx.scene.control.ToggleButton -selected: BooleanProperty -toggleGroup: ObjectProperty<ToggleGroup> +ToggleButton() +ToggleButton(text: String) +ToggleButton(text: String, graphic: Node)

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.

Indicates whether the button is selected.

Specifies the button group to which the button belongs.

Creates an empty toggle button.

Creates a toggle button with the specified text.

Creates a toggle button with the specified text and graphic.

javafx.scene.control.RadioButton

- +RadioButton()
- +RadioButton(text: String)

Creates an empty radio button.

Creates a radio button with the specified text.

ToggleButton and **RadioButton** are specialized buttons for making selections. **FIGURE 16.9**

RadioButton

```
RadioButton rbUS = new RadioButton("US");
rbUS.setGraphic(new ImageView("image/usIcon.gif"));
rbUS.setTextFill(Color.GREEN);
rbUS.setContentDisplay(ContentDisplay.LEFT);
rbUS.setStyle("-fx-border-color: black");
rbUS.setSelected(true);
rbUS.setPadding(new Insets(5, 5, 5,));
```



To group radio buttons, you need to create an instance of ToggleGroup and set a radio button's **toggleGroup** property to join the group, as follows:

```
ToggleGroup group = new ToggleGroup();
rbRed.setToggleGroup(group);
rbGreen.setToggleGroup(group);
rbBlue.setToggleGroup(group);
```

See Code: RadioButtonDemo.java

TextField

javafx.scene.control.TextInputControl -text: StringProperty -editable: BooleanProperty javafx.scene.control.TextField -alignment: ObjectProperty<Pos> -prefColumnCount: IntegerProperty -onAction: ObjectProperty<EventHandler<ActionEvent>> +TextField() +TextField(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 text content of this control.

Indicates whether the text can be edited by the user.

Specifies how the text should be aligned in the text field.

Specifies the preferred number of columns in the text field.

Specifies the handler for processing the action event on the text field.

Creates an empty text field.

Creates a text field with the specified text.

FIGURE 16.11 TextField enables the user to enter or display a string.

See TextFieldDemo.java

TextArea

javafx.scene.control.TextInputControl

-text: StringProperty

-editable: BooleanProperty

javafx.scene.control.TextArea

-prefColumnCount: IntegerProperty

-prefRowCount: IntegerProperty

-wrapText: BooleanProperty

+TextArea()

+TextArea(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 text content of this control.

Indicates whether the text can be edited by the user.

Specifies the preferred number of text columns.

Specifies the preferred number of text rows.

Specifies whether the text is wrapped to the next line.

Creates an empty text area.

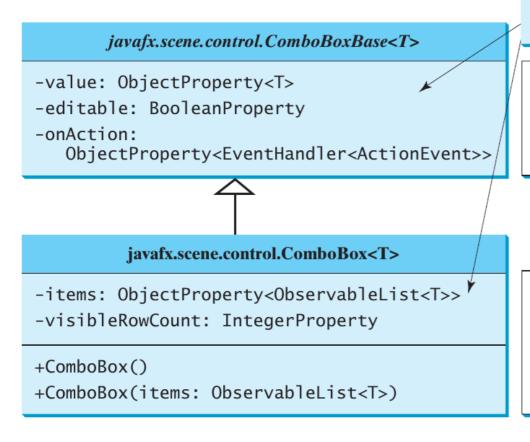
Creates a text area with the specified text.

FIGURE 16.13 TextArea enables the user to enter or display multiple lines of characters.

See Code: TextAreaDemo.java

ComboBox

- Presents a drop-down list of items the user can choose from.
- Can fire an ActionEvent whenever an item is selected.
- See Code: ComboBoxDemo.java



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 value selected in the combo box.

Specifies whether the combo box allows user input.

Specifies the handler for processing the action event.

The items in the combo box popup.

The maximum number of visible rows of the items in the combo box popup.

Creates an empty combo box.

Creates a combo box with the specified items.

FIGURE 16.16 ComboBox enables the user to select an item from a list of items.

ListView

- Same functionality as a ComboBox, but allows the user to choose one or more values.
- See ListViewDemo.java

javafx.scene.control.ListView<T>

-items: ObjectProperty<ObservableList<T>>

-orientation: BooleanProperty

-selectionModel:

ObjectProperty<MultipleSelectionModel<T>>

+ListView()

+ListView(items: ObservableList<T>)

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 items in the list view.

Indicates whether the items are displayed horizontally or vertically in the list view.

Specifies how items are selected. The SelectionModel is also used to obtain the selected items.

Creates an empty list view.

Creates a list view with the specified items.

FIGURE 16.18 ListView enables the user to select one or multiple items from a list of items.

ScrollBar

See Code: ScrollBarDemo.java

javafx.scene.control.ScrollBar

-blockIncrement: DoubleProperty

-max: DoubleProperty

-min: DoubleProperty

-unitIncrement: DoubleProperty

-value: DoubleProperty

-visibleAmount: DoubleProperty

-orientation: ObjectProperty<Orientation>

+ScrollBar()

+increment()

+decrement()

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 amount to adjust the scroll bar if the track of the bar is clicked (default: 10).

The maximum value represented by this scroll bar (default: 100).

The minimum value represented by this scroll bar (default: 0).

The amount to adjust the scroll bar when the increment() and decrement() methods are called (default: 1).

Current value of the scroll bar (default: 0).

The width of the scroll bar (default: 15).

Specifies the orientation of the scroll bar (default: HORIZONTAL).

Creates a default horizontal scroll bar.

Increments the value of the scroll bar by unitIncrement.

Decrements the value of the scroll bar by unitIncrement.

FIGURE 16.22 ScrollBar enables the user to select from a range of values.

Slider

- Similar to a ScrollBar but has more properties and can appear in many forms.
- See Code:
 - SliderDemo.java
 - BounceBallSlider.java

javafx.scene.control.Slider

-blockIncrement: DoubleProperty

-max: DoubleProperty
-min: DoubleProperty

-value: DoubleProperty

-orientation: ObjectProperty<Orientation>

-majorTickUnit: DoubleProperty

-minorTickCount: IntegerProperty

-showTickLabels: BooleanProperty

-showTickMarks: BooleanProperty

+Slider()

+Slider(min: double, max: double,

value: 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 amount to adjust the slider if the track of the bar is clicked (default: 10).

The maximum value represented by this slider (default: 100).

The minimum value represented by this slider (default: 0).

Current value of the slider (default: 0).

Specifies the orientation of the slider (default: HORIZONTAL).

The unit distance between major tick marks.

The number of minor ticks to place between two major ticks.

Specifies whether the labels for tick marks are shown.

Specifies whether the tick marks are shown.

Creates a default horizontal slider.

Creates a slider with the specified min, max, and value.

FIGURE 16.25 Slider enables the user to select from a range of values.

Tic-Tac-Toe Game

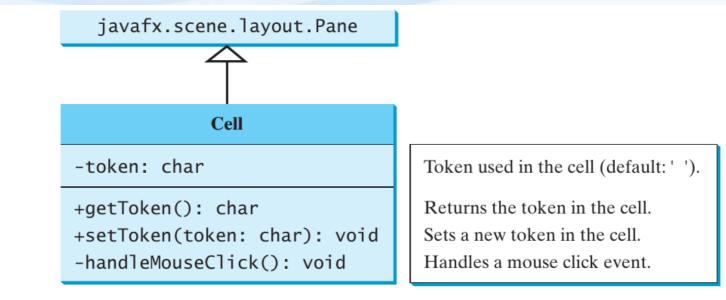


FIGURE 16.28 The Cell class displays the token in a cell.

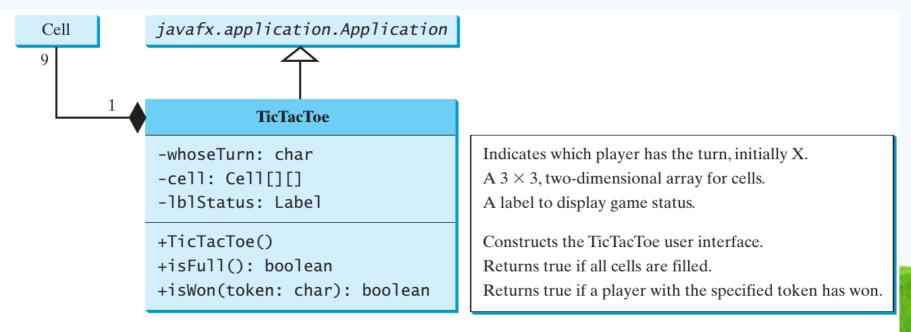


FIGURE 16.29 The TicTacToe class contains nine cells.