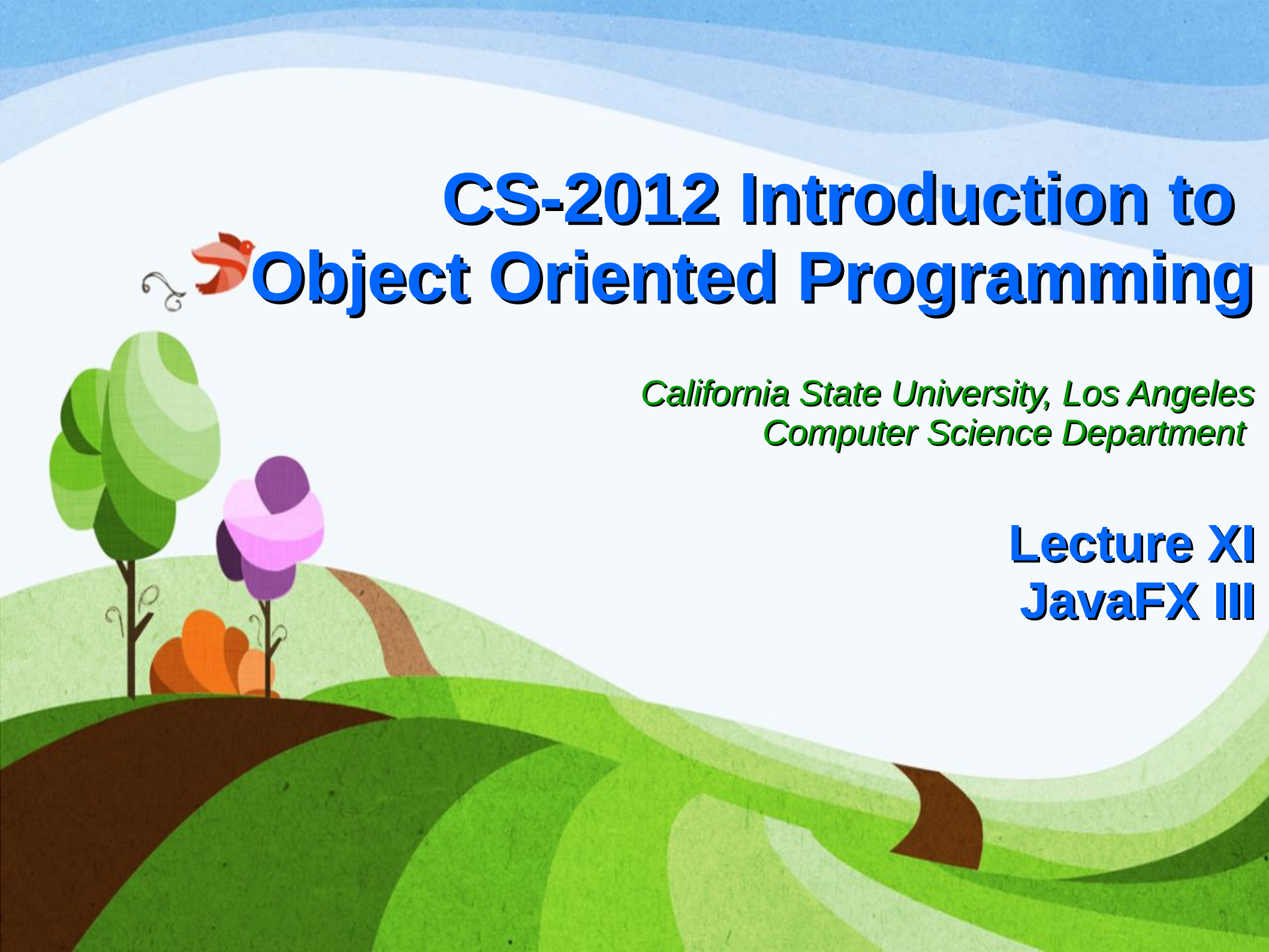
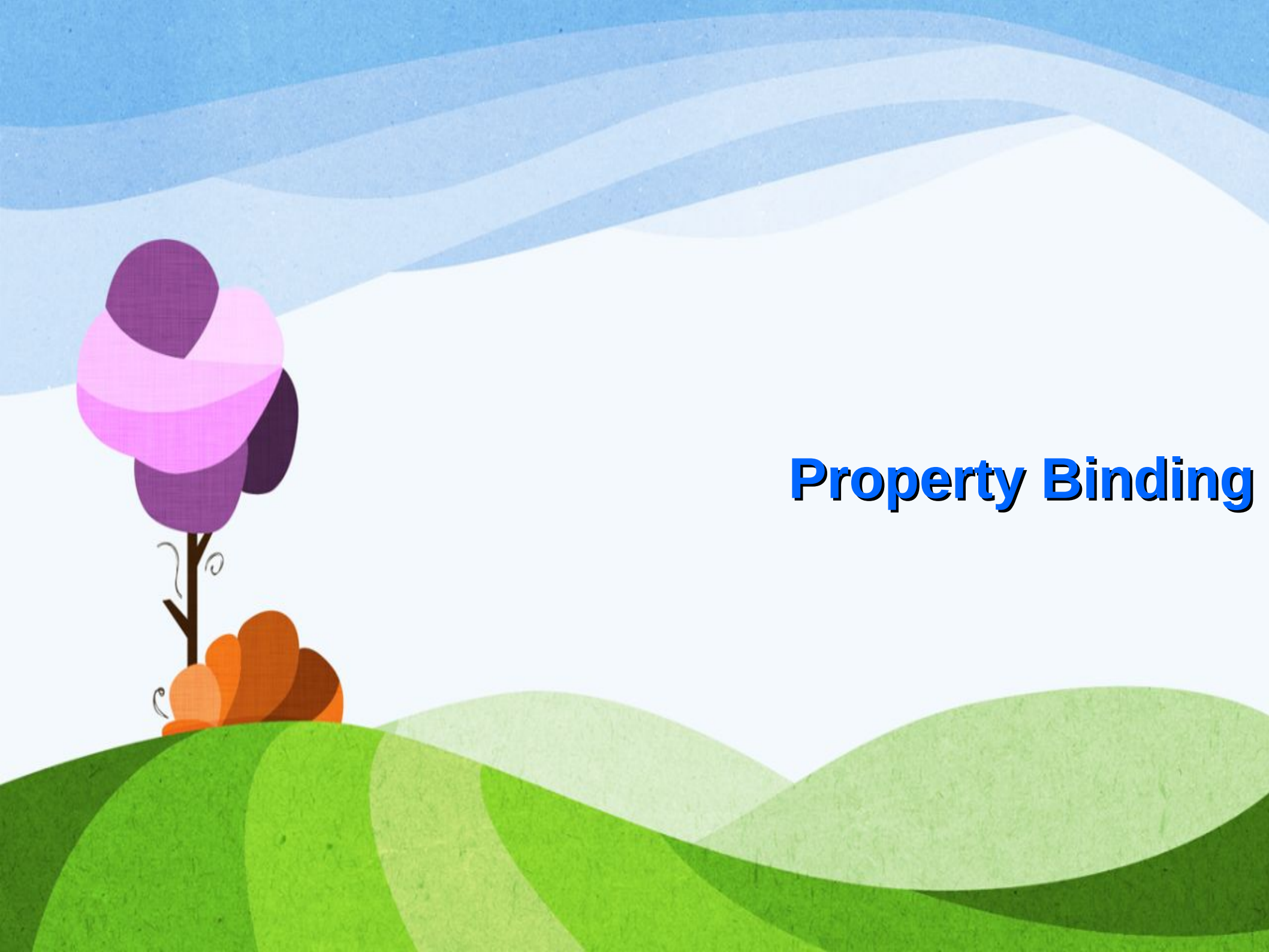


CS-2012 Introduction to Object Oriented Programming

California State University, Los Angeles
Computer Science Department

Lecture XI
JavaFX III


Property Binding

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

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.layout.Pane;
4  import javafx.scene.paint.Color;
5  import javafx.scene.shape.Circle;
6  import javafx.stage.Stage;
7
8  public class ShowCircleCentered extends Application {
9    @Override // Override the start method in the Application class
10   public void start(Stage primaryStage) {
11     // Create a pane to hold the circle
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() { ... }
}
```

(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

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

BindingDemo.java

LISTING 14.6 BindingDemo.java

```
1  import javafx.beans.property.DoubleProperty;
2  import javafx.beans.property.SimpleDoubleProperty;
3
4  public class BindingDemo {
5    public static void main(String[] args) {
6      DoubleProperty d1 = new SimpleDoubleProperty(1);
7      DoubleProperty d2 = new SimpleDoubleProperty(2);
8      d1.bind(d2);
9      System.out.println("d1 is " + d1.getValue()
10       + " and d2 is " + d2.getValue());
11     d2.setValue(70.2);
12     System.out.println("d1 is " + d1.getValue()
13       + " and d2 is " + d2.getValue());
14   }
15 }
```

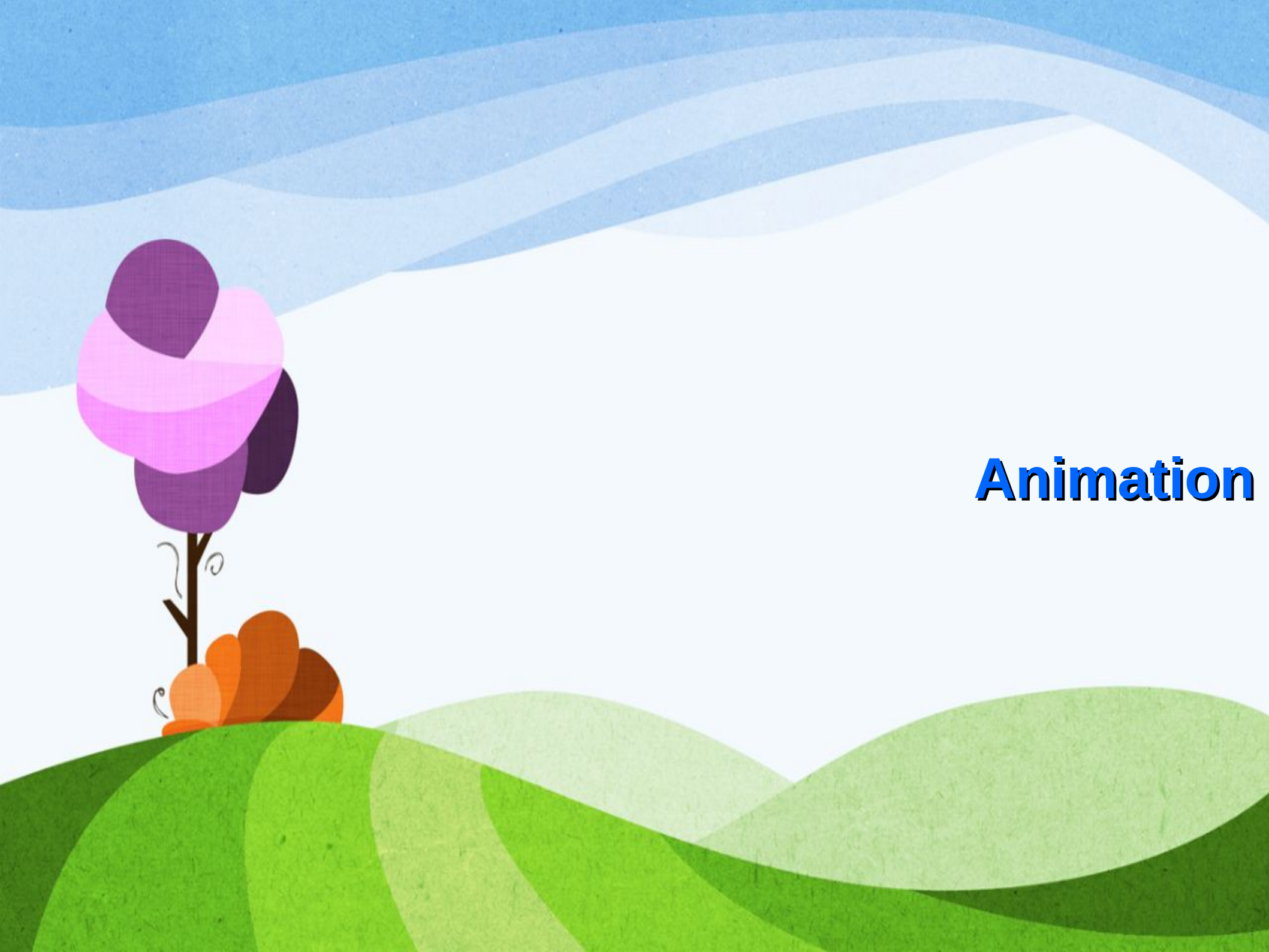
BindingDemo.java

- 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

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`**

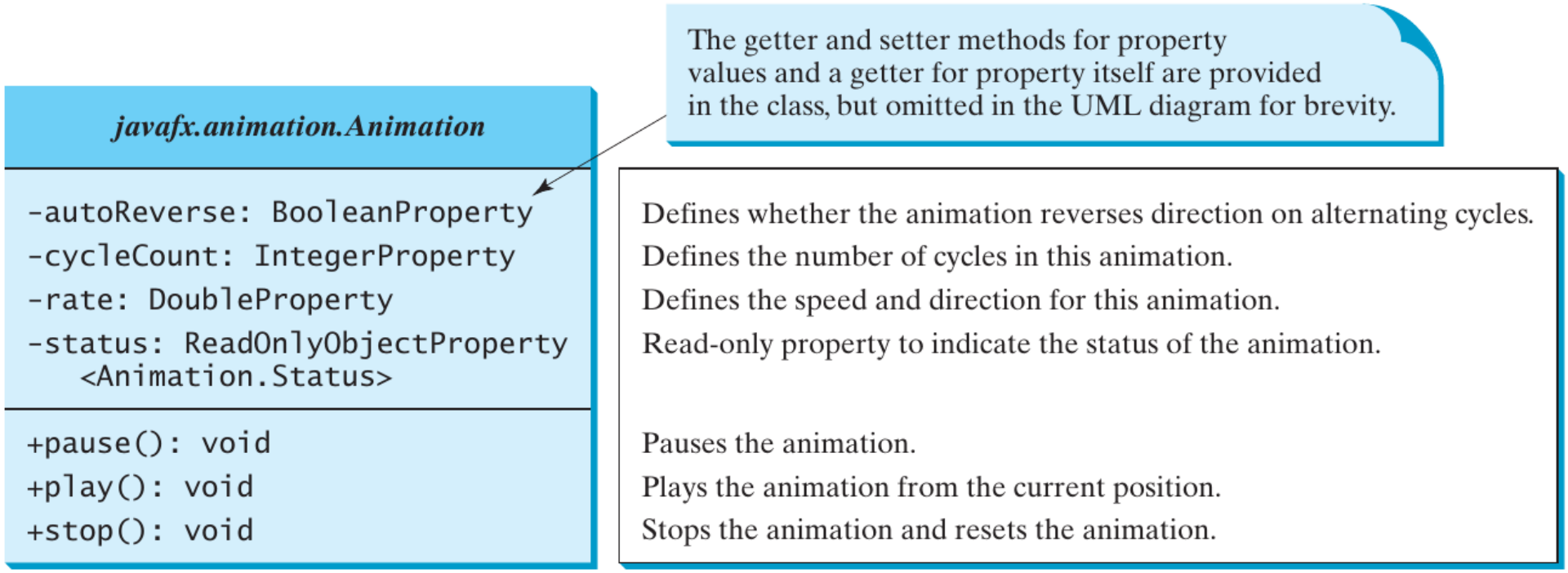


FIGURE 15.15 The abstract `Animation` class is the root class for JavaFX animations.

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.

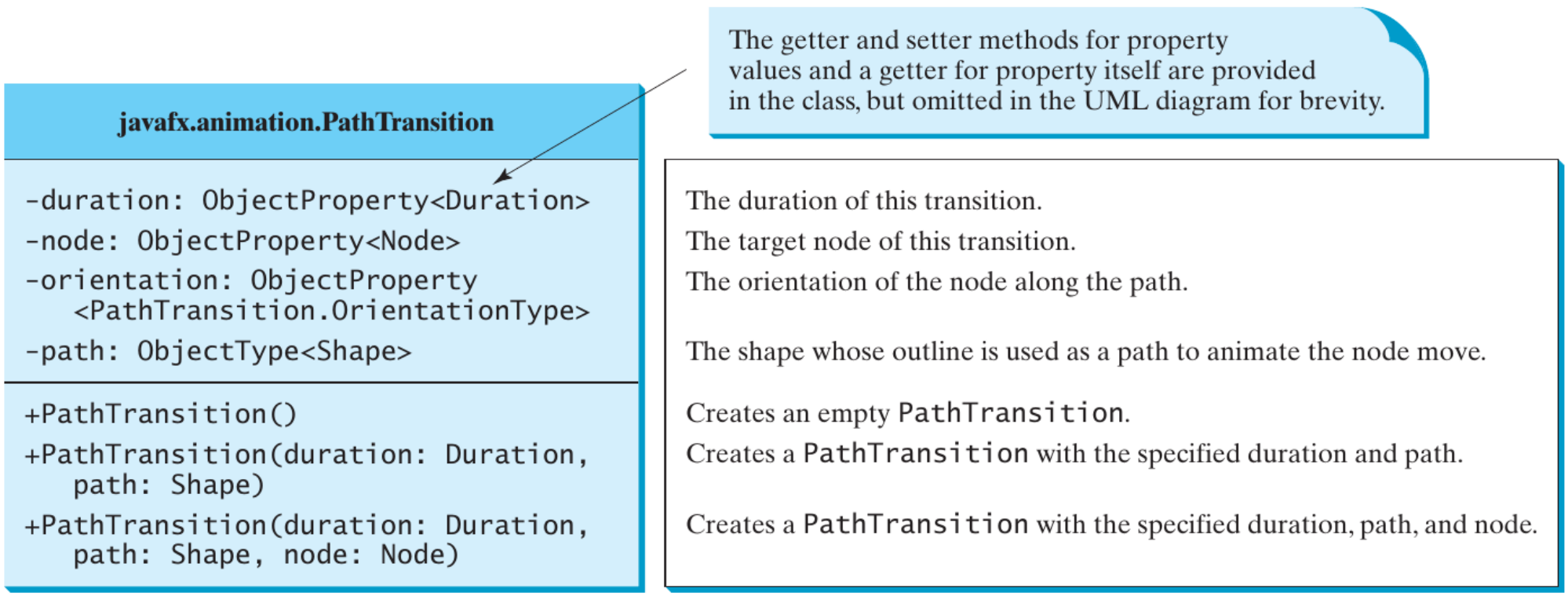


FIGURE 15.16 The `PathTransition` class defines an animation for a node along a path.

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.

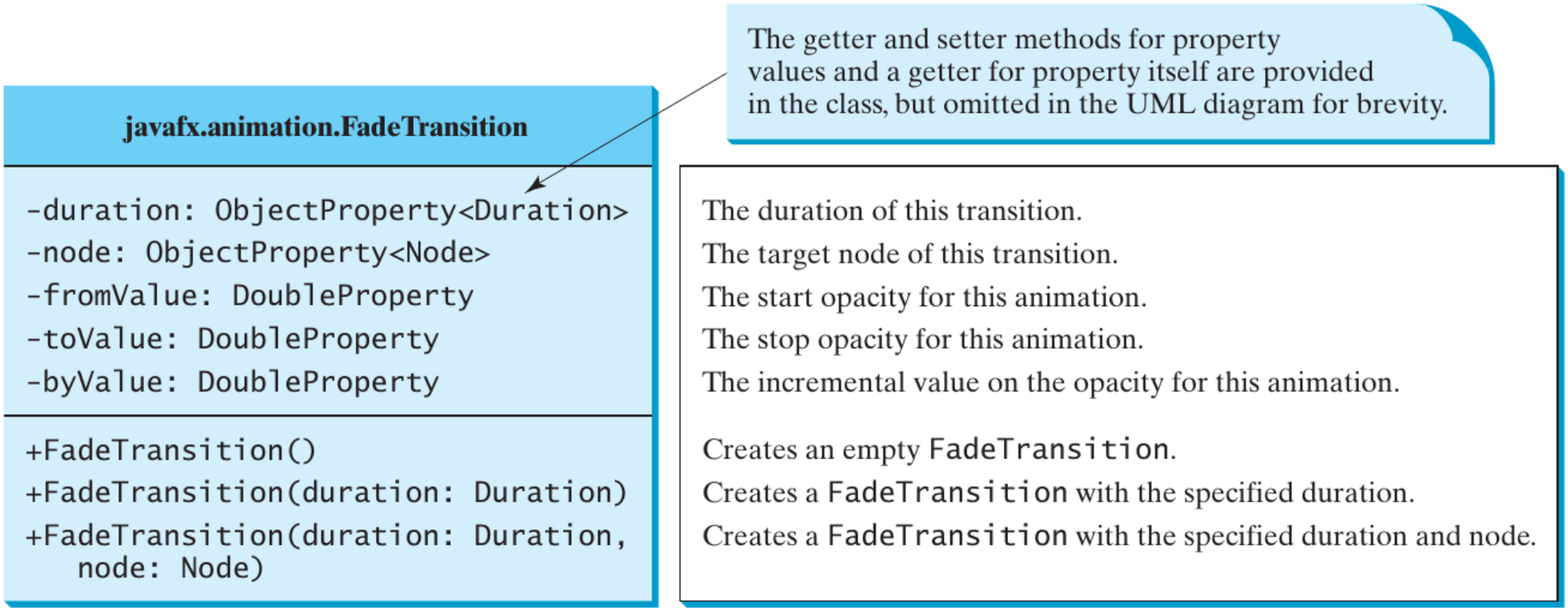


FIGURE 15.18 The `FadeTransition` class defines an animation for the change of opacity in a node.

- See Code: **`FadeTransitionDemo.java`**

TimeLine

- Used to program any animation using one or more **`KeyFrame`**s.
- 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`**

JavaFX UI Controls

Common JavaFX UI Controls

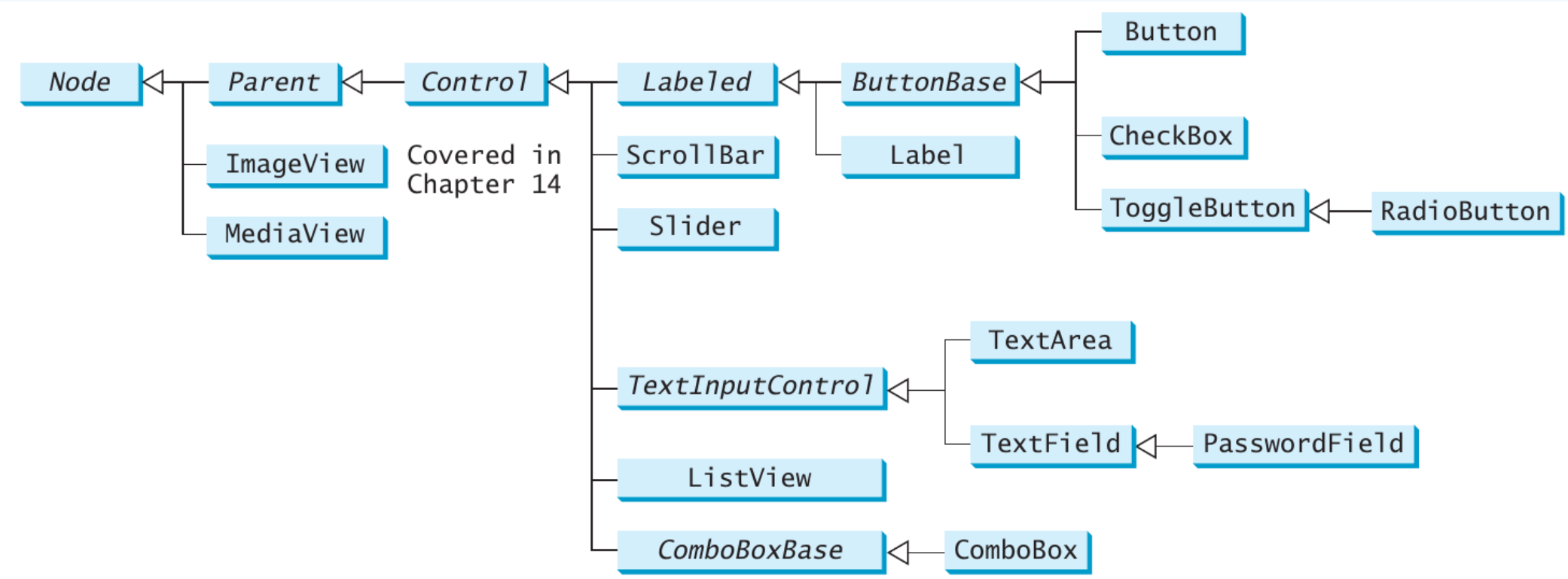


FIGURE 16.1 These UI controls are frequently used to create user interfaces.

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.

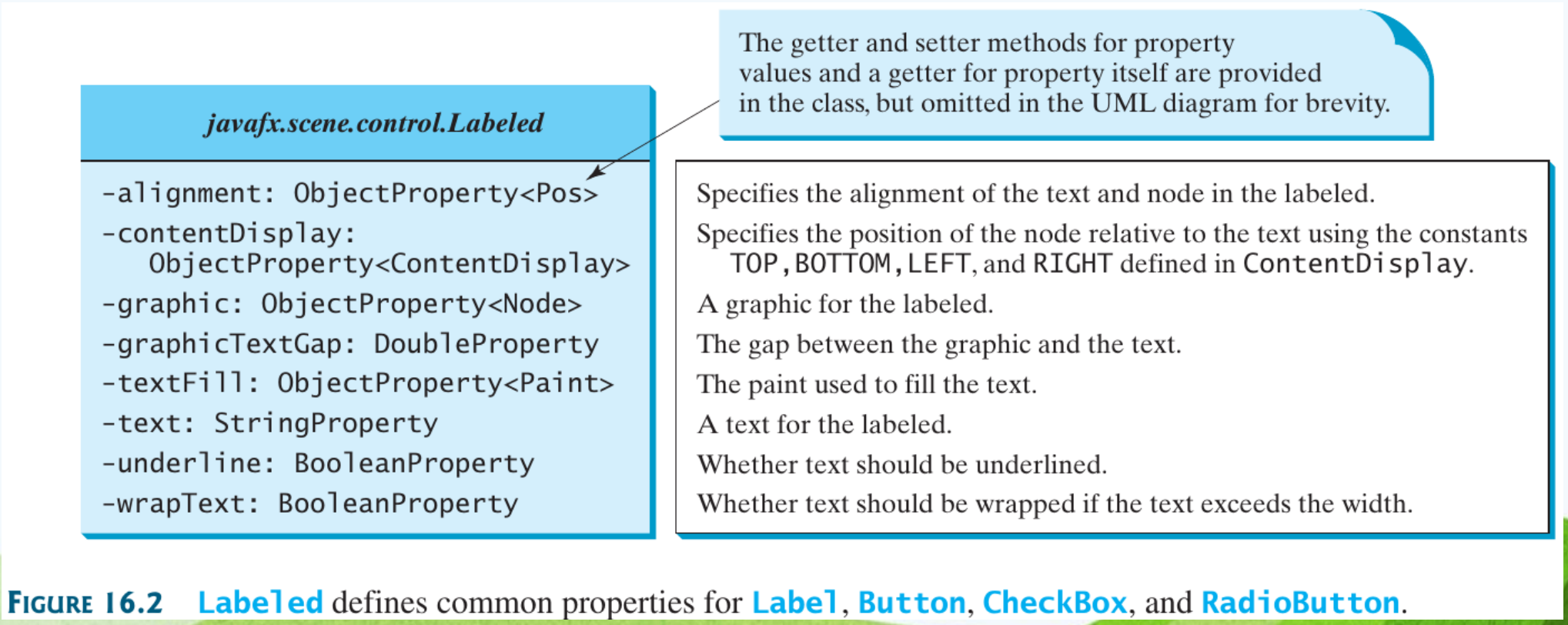


FIGURE 16.2 `Labeled` defines common properties for `Label`, `Button`, `CheckBox`, and `RadioButton`.

Labeled and Label

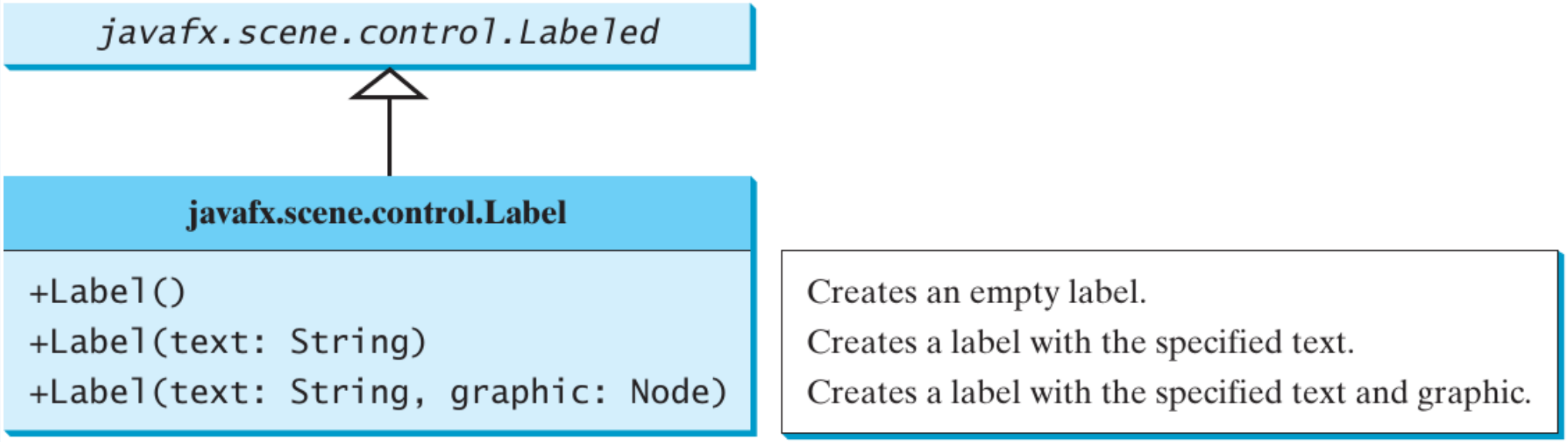


FIGURE 16.3 `Label` is created to display a text or a node, or both.

- See Code: **`LabelWithGraphic.java`**

Button

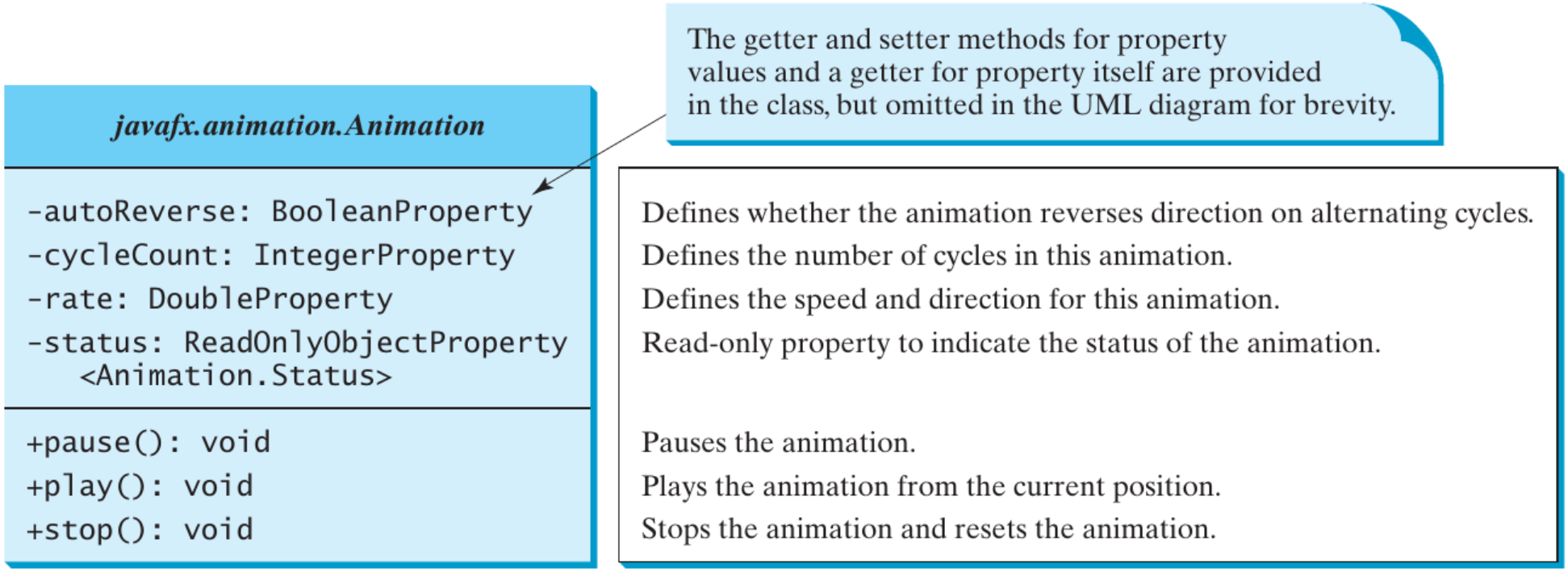


FIGURE 15.15 The abstract `Animation` class is the root class for JavaFX animations.

- 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`**

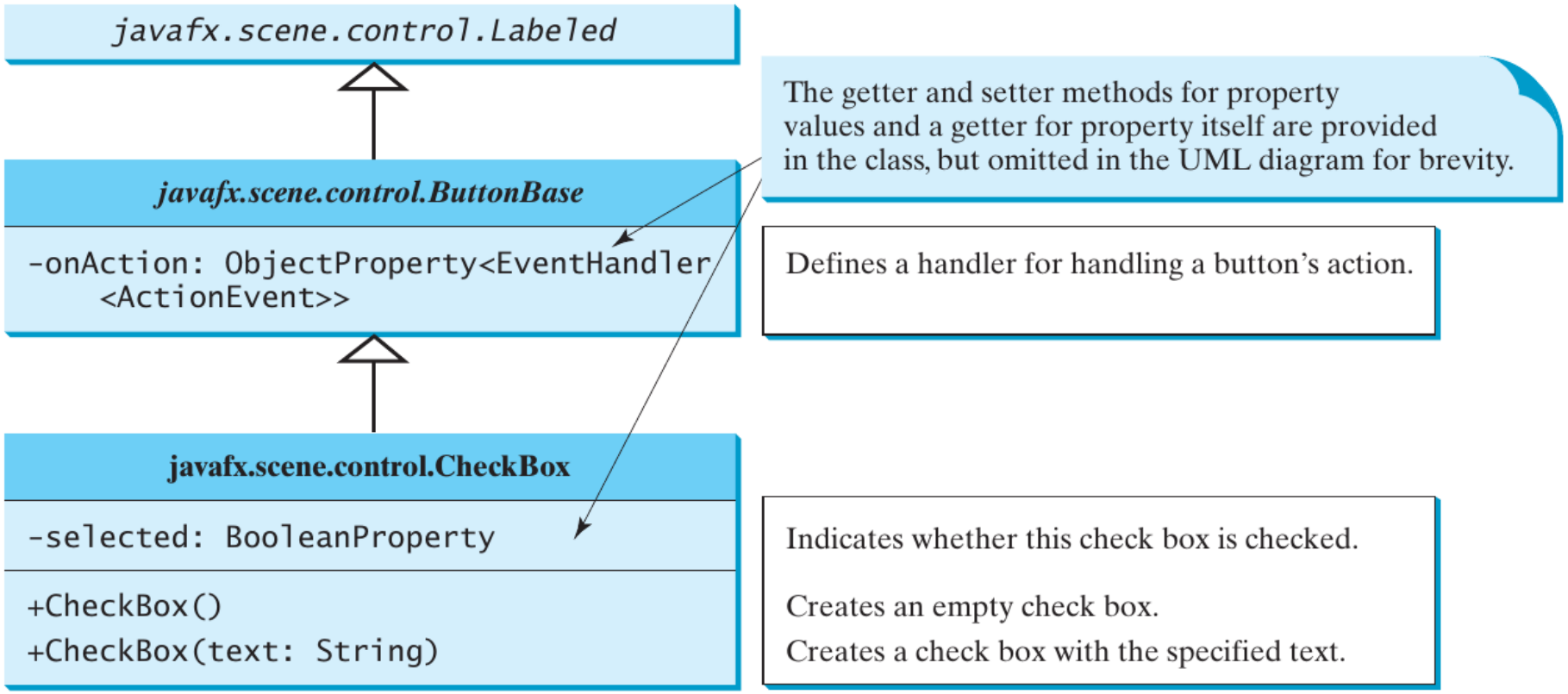


FIGURE 16.7 CheckBox contains the properties inherited from ButtonBase and Labeled.

RadioButton

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

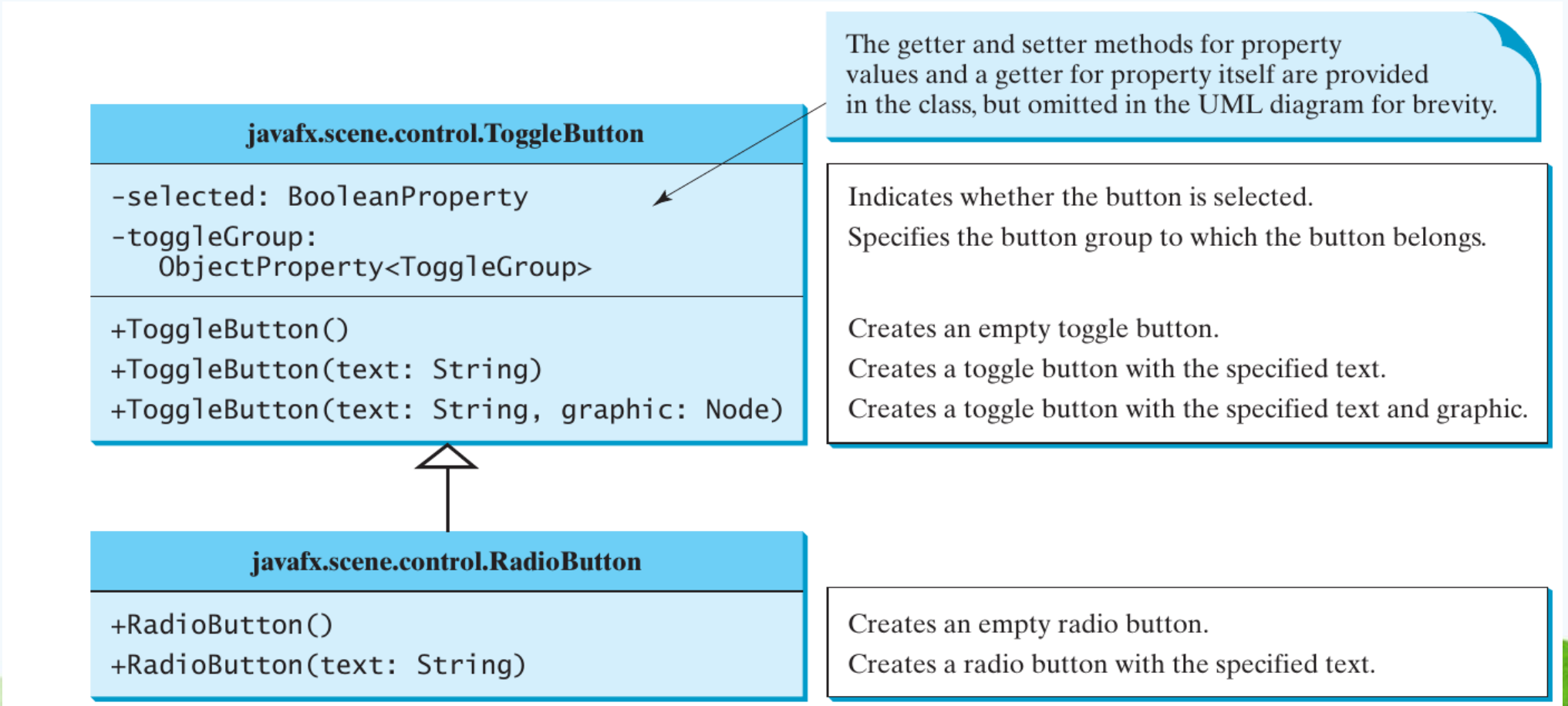


FIGURE 16.9 `ToggleButton` and `RadioButton` are specialized buttons for making selections.

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

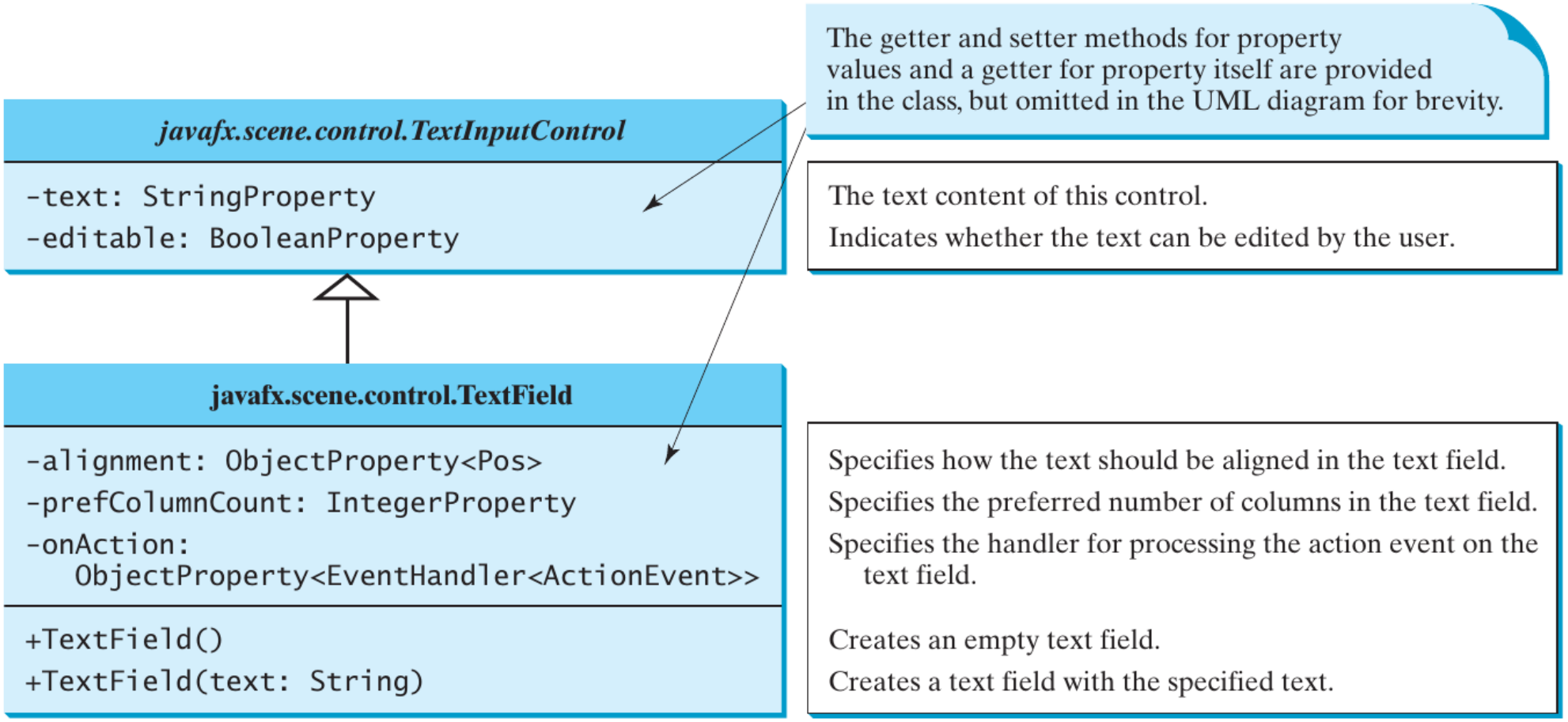


FIGURE 16.11 `TextField` enables the user to enter or display a string.

- See **`TextFieldDemo.java`**

TextArea

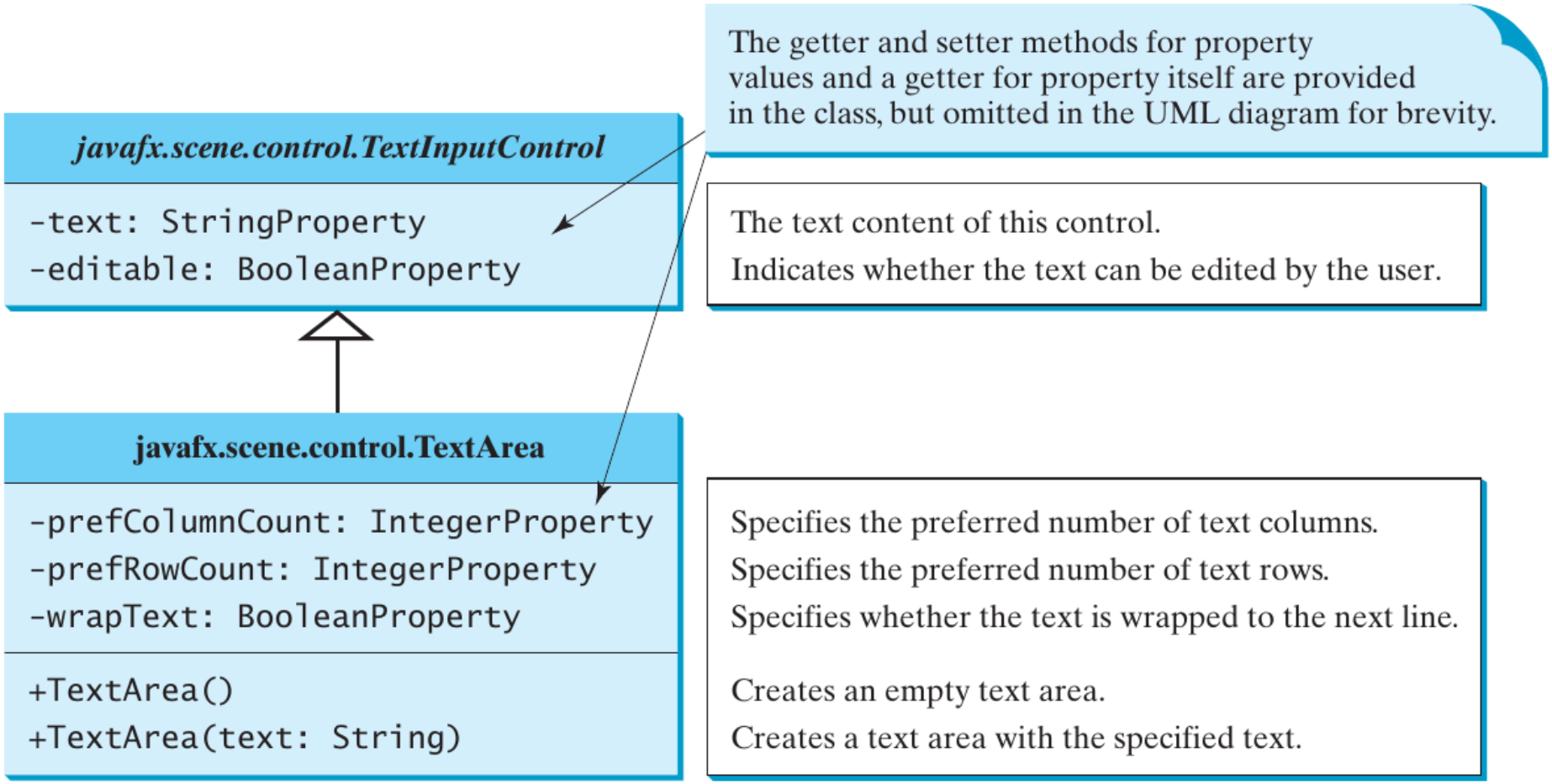


FIGURE 16.13 `TextArea` enables the user to enter or display multiple lines of characters.

- 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`**

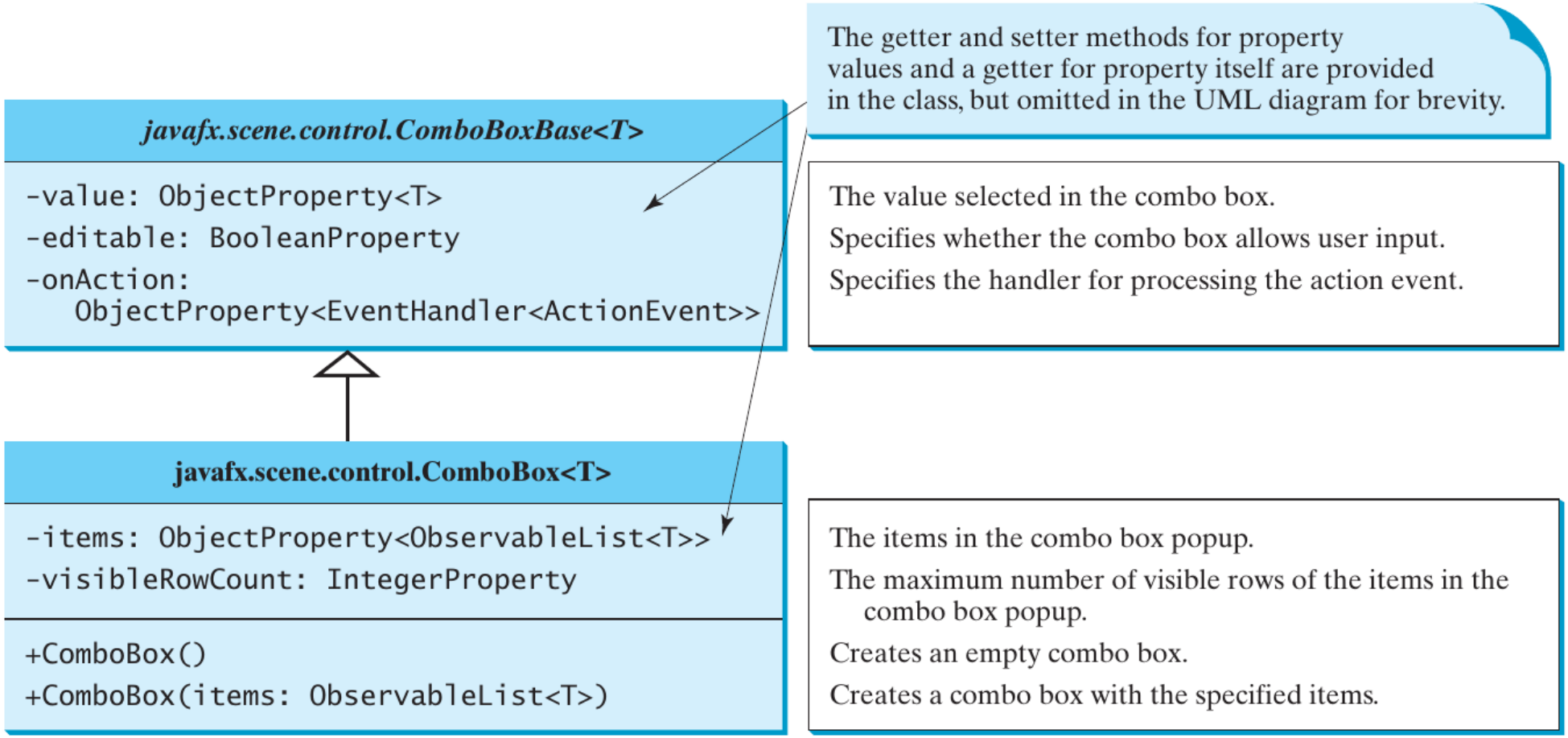


FIGURE 16.16 ComboBox enables the user to select an item from a list of items.

ListView

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

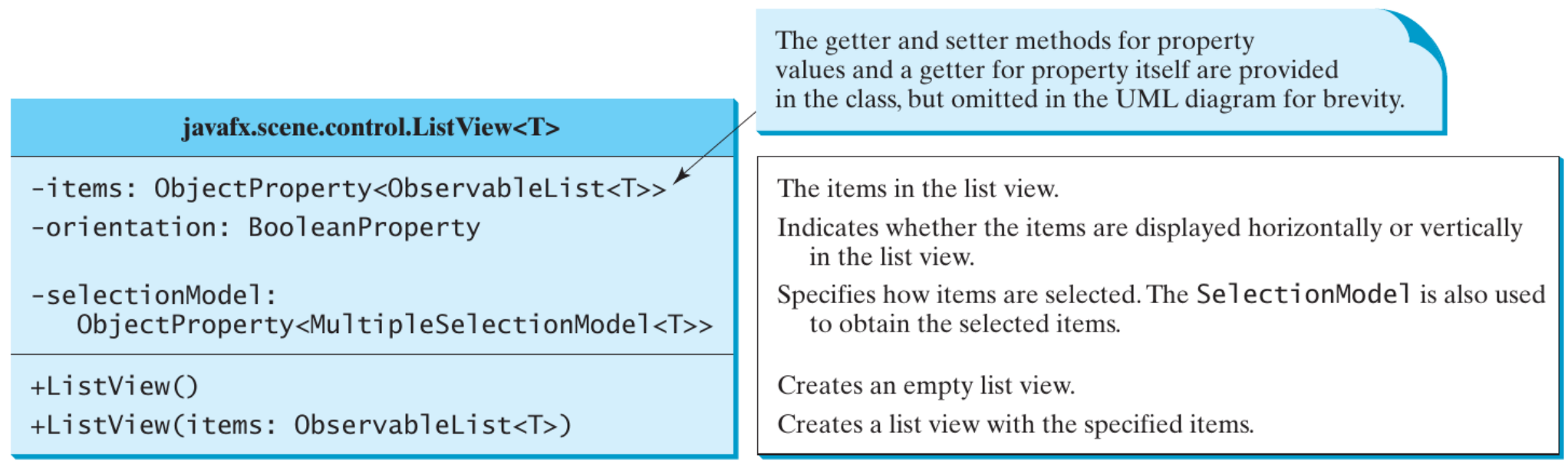


FIGURE 16.18 `ListView` enables the user to select one or multiple items from a list of items.

ScrollBar

- See Code: **`ScrollBarDemo.java`**

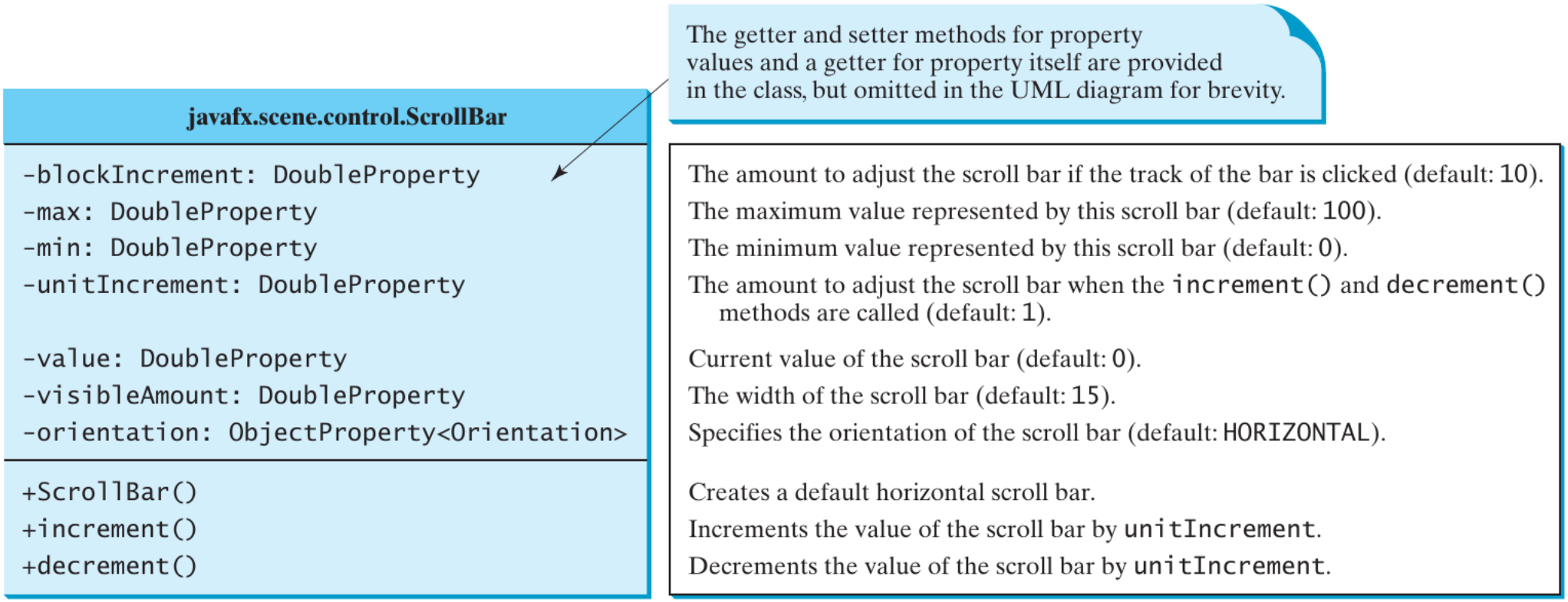


FIGURE 16.22 `ScrollBar` enables the user to select from a range of values.

Slider

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

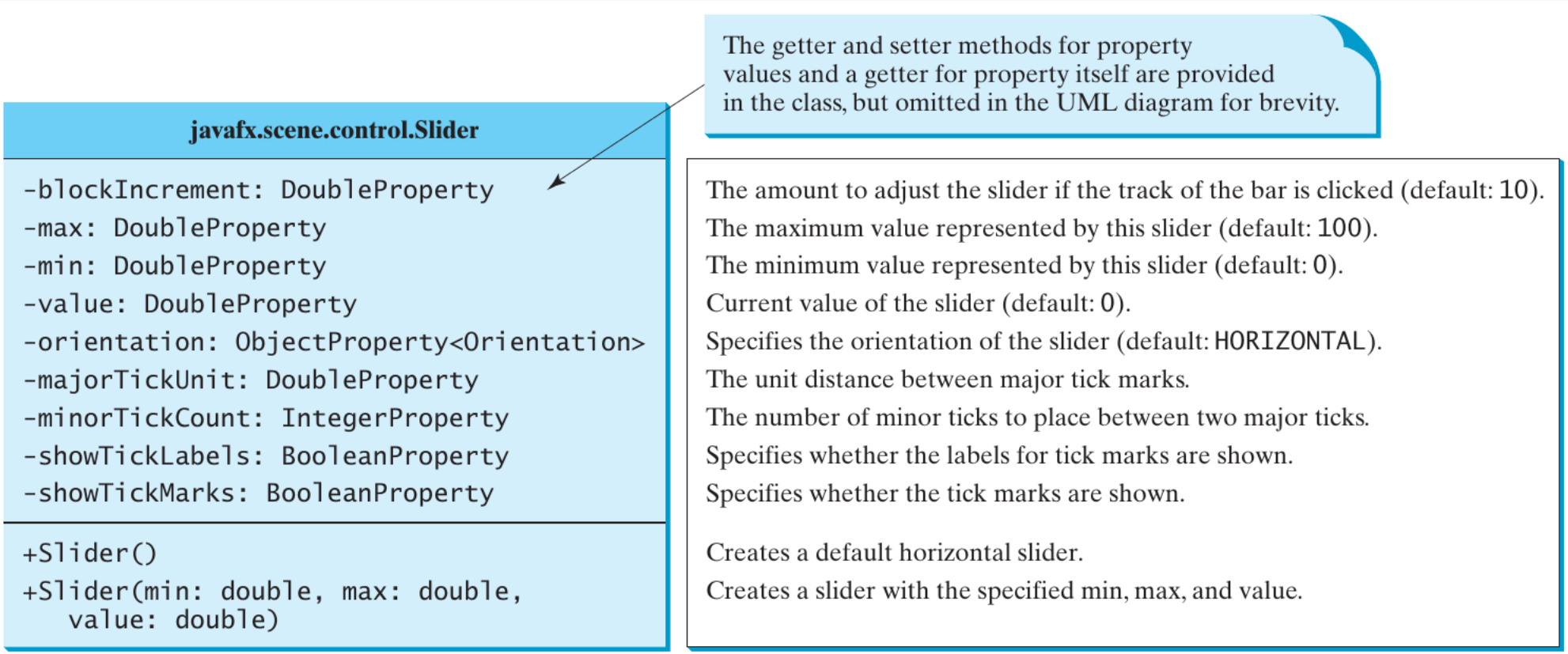


FIGURE 16.25 `Slider` enables the user to select from a range of values.

Tic-Tac-Toe Game

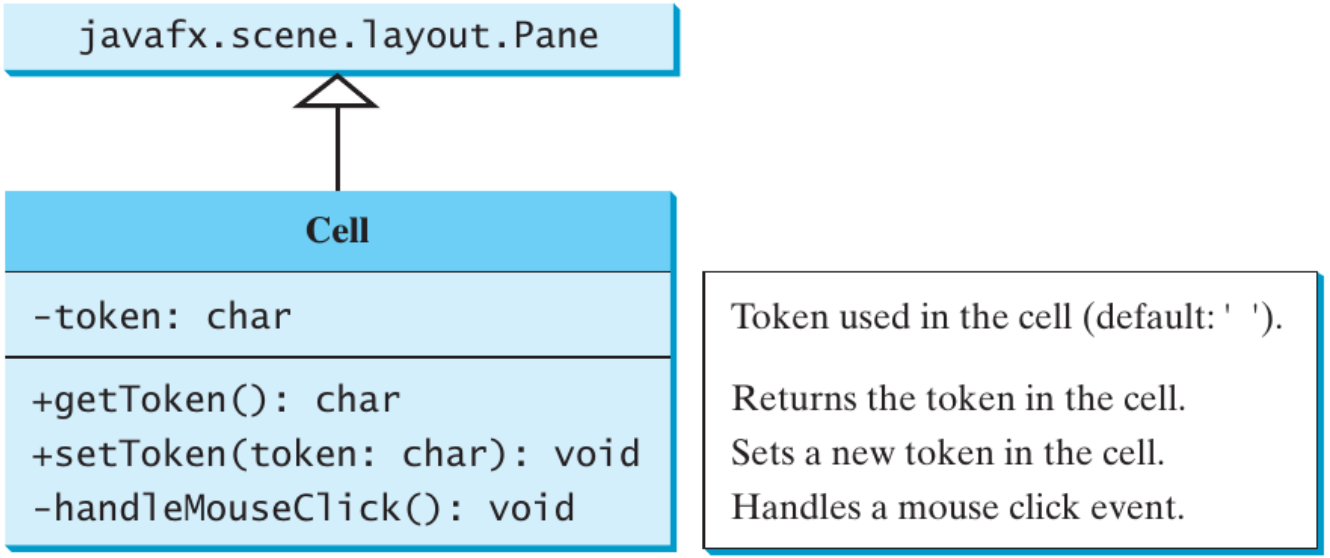


FIGURE 16.28 The `Cell` class displays the token in a cell.

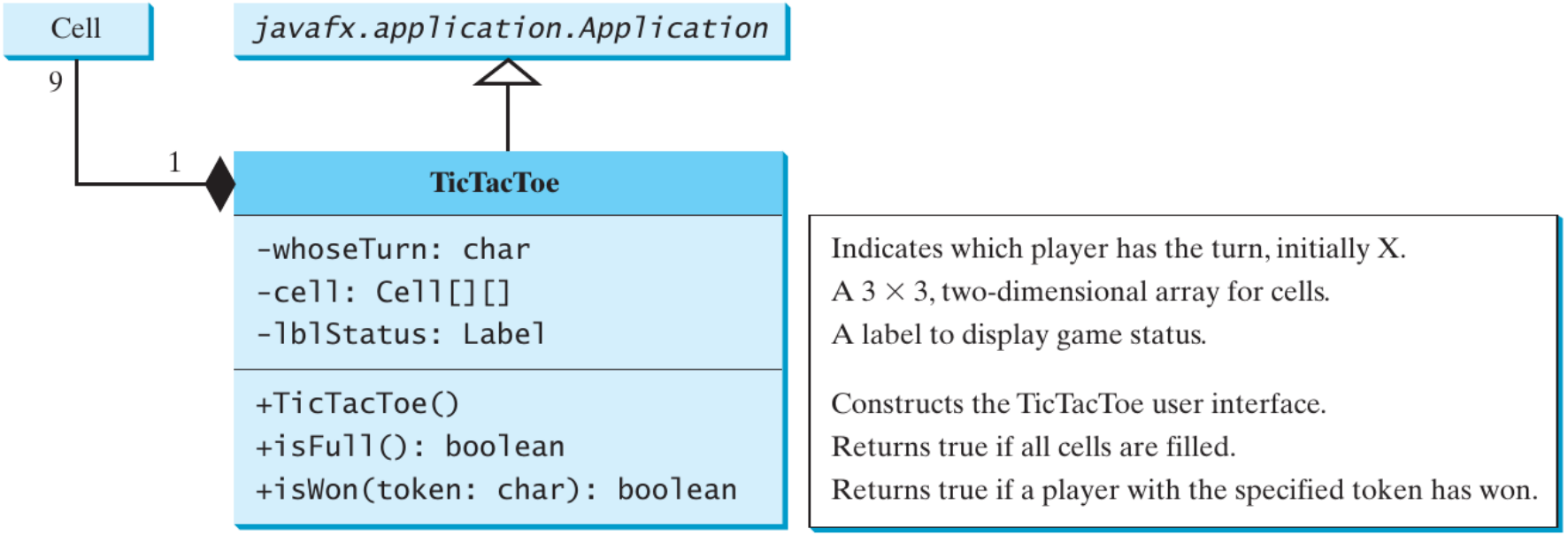


FIGURE 16.29 The `TicTacToe` class contains nine cells.