CS-2012 Introduction to Object Oriented Programming

California State University, Los Angeles
Computer Science Department

Lecture XI
JavaFX III

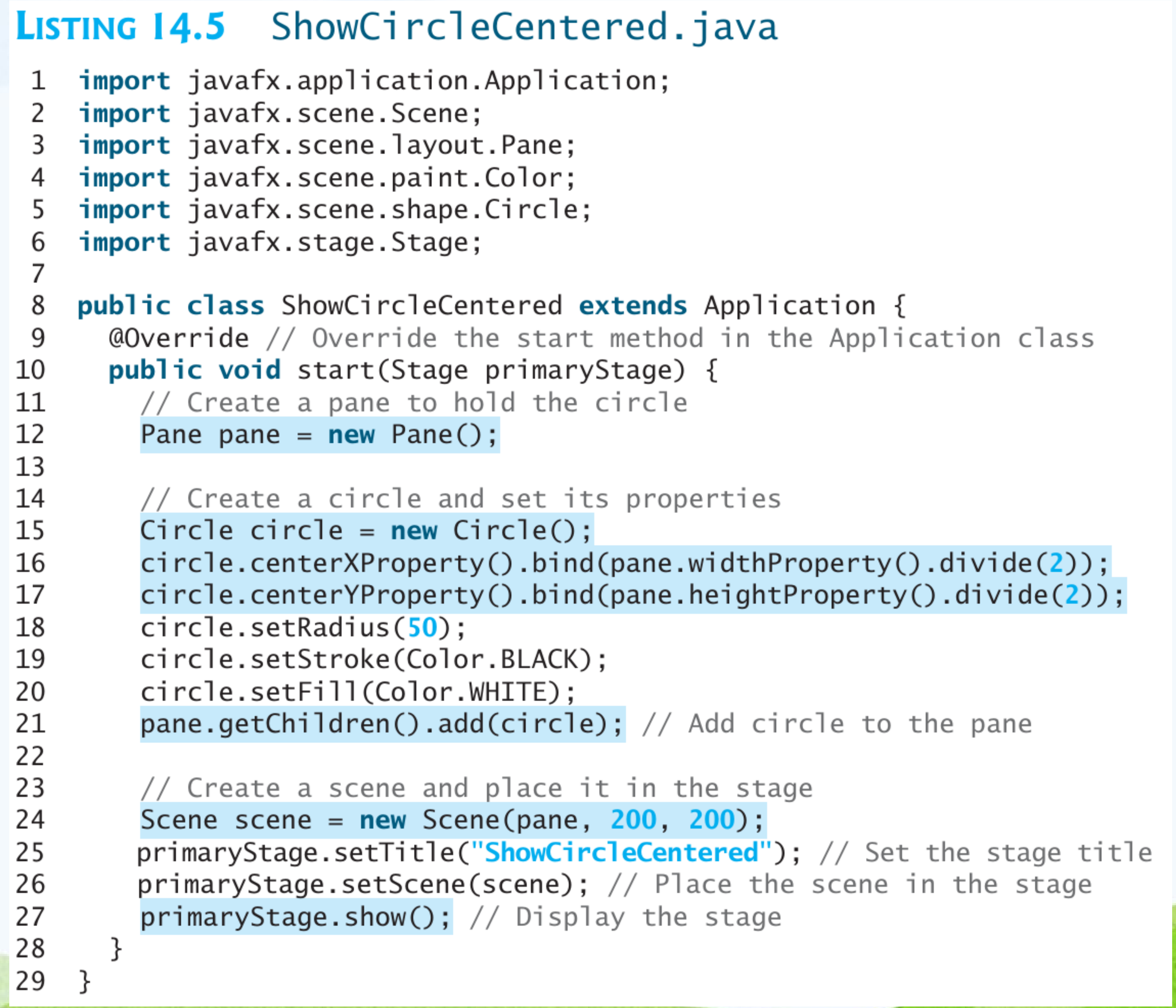
Property Binding

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

```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 {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane to hold the circle
    Pane pane = new Pane();

    // Create a circle and set its properties
    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

    // Create a scene and place it in the stage
    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
  }
}
```

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

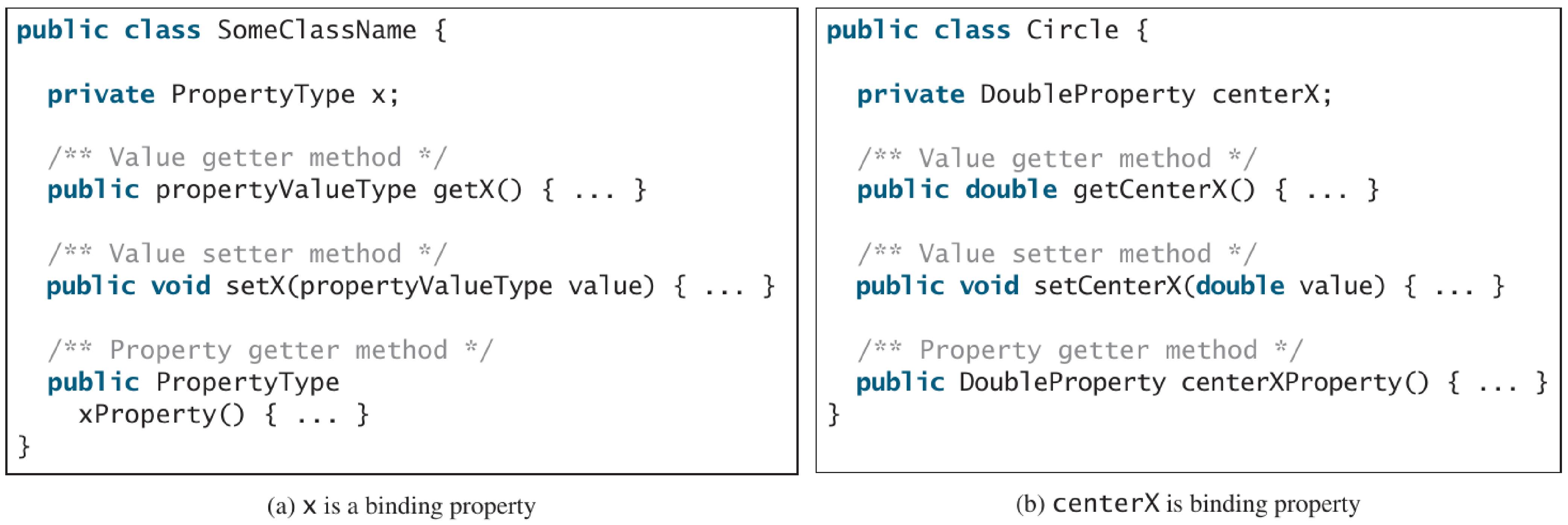
- **`DoubleProperty`**, **`FloatProperty`**, **`LongProperty`**, **`IntegerProperty`**, **`BooleanProperty`**, **`StringProperty`**
  - these are all subtypes of **`ObservableValue`** so they can also be used as source objects for binding

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() { ... }
}
```

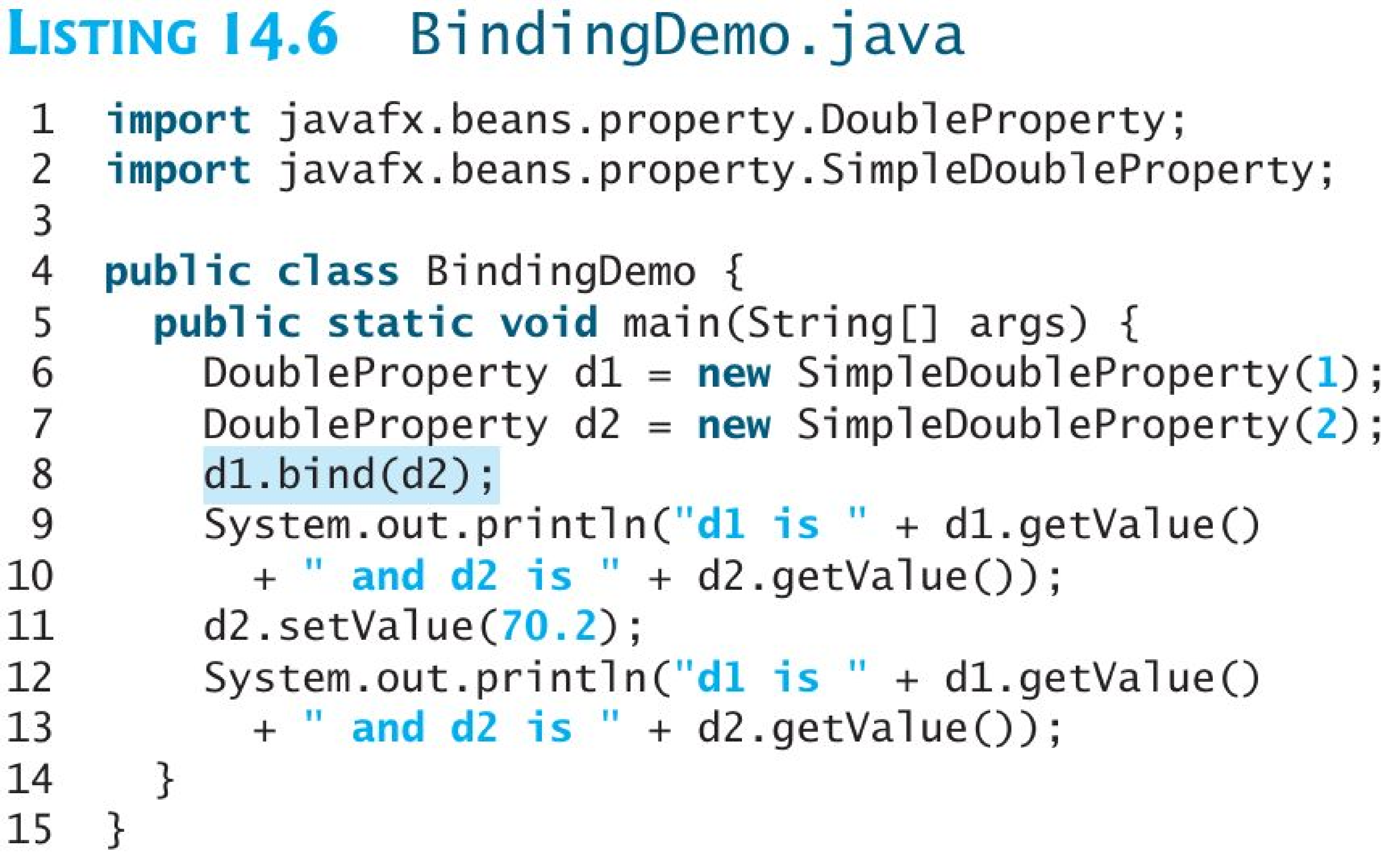
(a) x is a binding property

```
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

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

public class BindingDemo {
  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());
    d2.setValue(70.2);
    System.out.println("d1 is " + d1.getValue()
      + " and d2 is " + d2.getValue());
  }
}
```

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

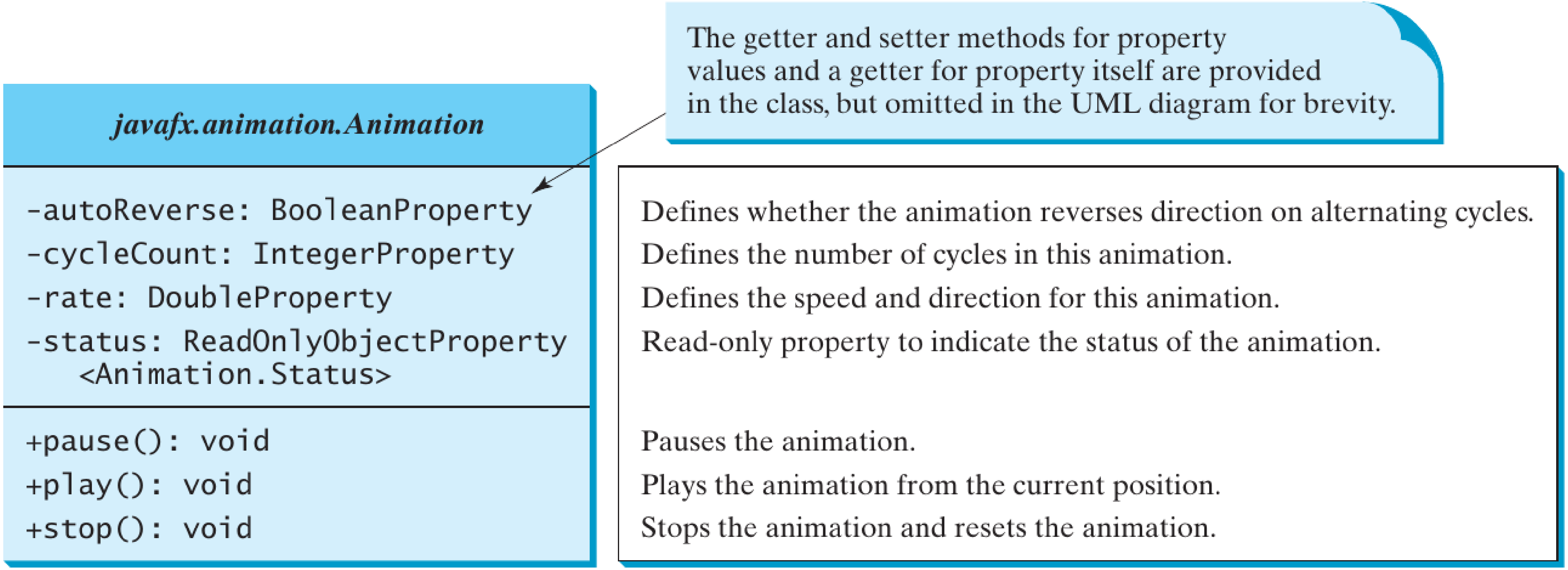
Animation

- 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`**



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.animation.Animation

```
-autoReverse: BooleanProperty
-cycleCount: IntegerProperty
-rate: DoubleProperty
-status: ReadOnlyObjectProperty
   <Animation.Status>

+pause(): void
+play(): void
+stop(): void
```

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.

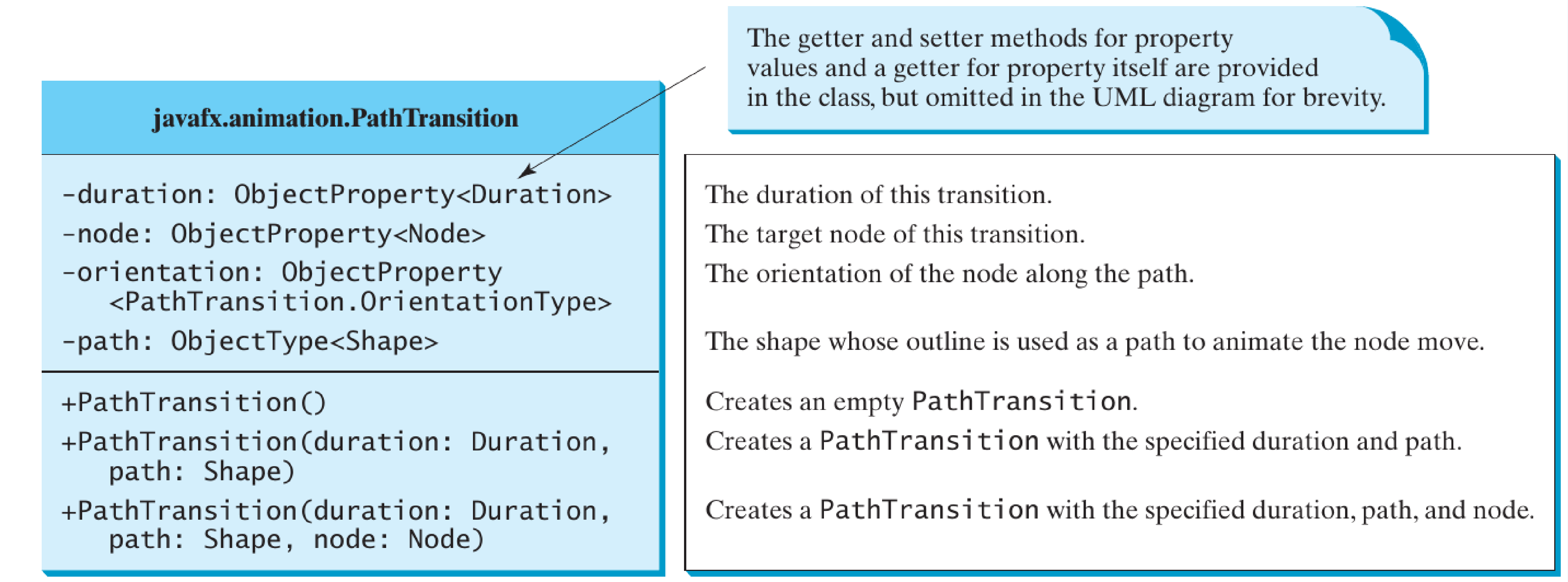
FIGURE 15.15 The abstract `Animation` class is the root class for JavaFX animations.

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.



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.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 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.

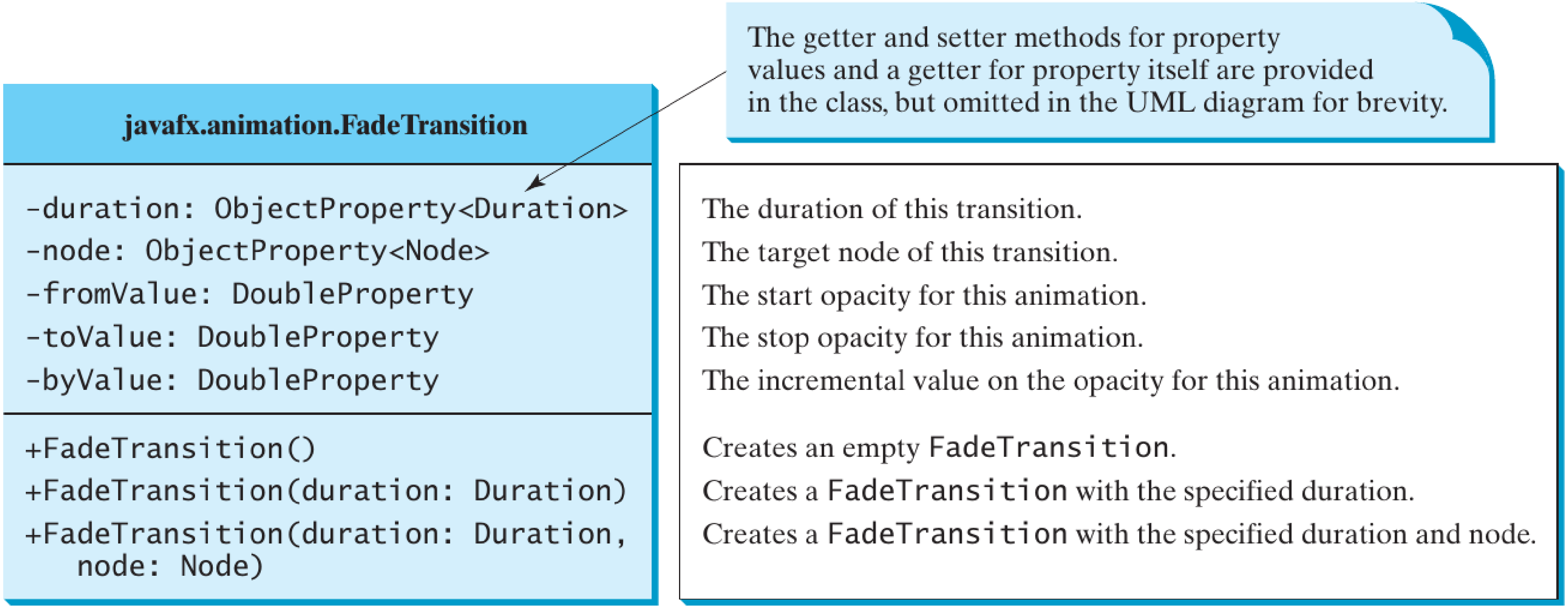
PathTransition

- **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.



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.animation.FadeTransition

`-duration: ObjectProperty<Duration>`	The duration of this transition.
`-node: ObjectProperty<Node>`	The target node of this transition.
`-fromValue: DoubleProperty`	The start opacity for this animation.
`-toValue: DoubleProperty`	The stop opacity for this animation.
`-byValue: DoubleProperty`	The incremental value on the opacity for this animation.
`+FadeTransition()`	Creates an empty `FadeTransition`.
`+FadeTransition(duration: Duration)`	Creates a `FadeTransition` with the specified duration.
`+FadeTransition(duration: Duration, node: Node)`	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`**