



COMP231

Advanced Programming

Chapter 15 Event-Driven Programming and Animations

Compiled By: Dr. Majdi Mafarja
Fall Semester 2017/2018

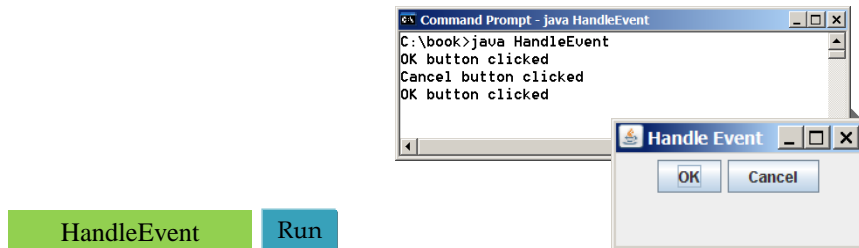
Procedural vs. Event-Driven Programming

- *Procedural programming* is executed in procedural order.
- In event-driven programming, code is executed upon activation of events.



Taste of Event-Driven Programming

The example displays a button in the frame. A message is displayed on the console when a button is clicked.



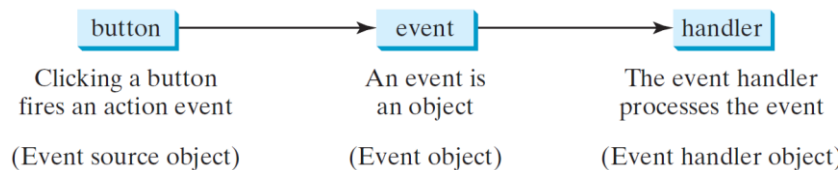
Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

3

Handling GUI Events

Source object (e.g., button)

Listener object contains a method for processing the event.



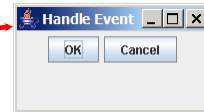
Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

4

Trace Execution

```
public class HandleEvent extends Application {
    public void start(Stage primaryStage) {
        ...
        OKHandlerClass handler1 = new OKHandlerClass();
        btOK.setOnAction(handler1);
        CancelHandlerClass handler2 = new CancelHandlerClass();
        btCancel.setOnAction(handler2);
        ...
        primaryStage.show(); // Display the stage
    }
}
```

1. Start from the main method to create a window and display it



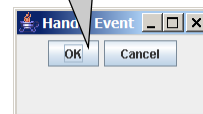
```
class OKHandlerClass implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        System.out.println("OK button clicked");
    }
}
```



Trace Execution

```
public class HandleEvent extends Application {
    public void start(Stage primaryStage) {
        ...
        OKHandlerClass handler1 = new OKHandlerClass();
        btOK.setOnAction(handler1);
        CancelHandlerClass handler2 = new CancelHandlerClass();
        btCancel.setOnAction(handler2);
        ...
        primaryStage.show(); // Display the stage
    }
}
```

2. Click OK



```
class OKHandlerClass implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        System.out.println("OK button clicked");
    }
}
```

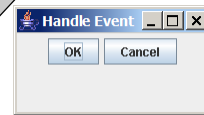


Trace Execution

```
public class HandleEvent extends Application {
    public void start(Stage primaryStage) {
        ...
        OKHandlerClass handler1 = new OKHandlerClass();
        btOK.setOnAction(handler1);
        CancelHandlerClass handler2 = new CancelHandlerClass();
        btCancel.setOnAction(handler2);
        ...
        primaryStage.show(); // Display the stage
    }
}
```

```
class OKHandlerClass implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        System.out.println("OK button clicked");
    }
}
```

3. The JVM invokes the listener's handle method



Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

7

Events

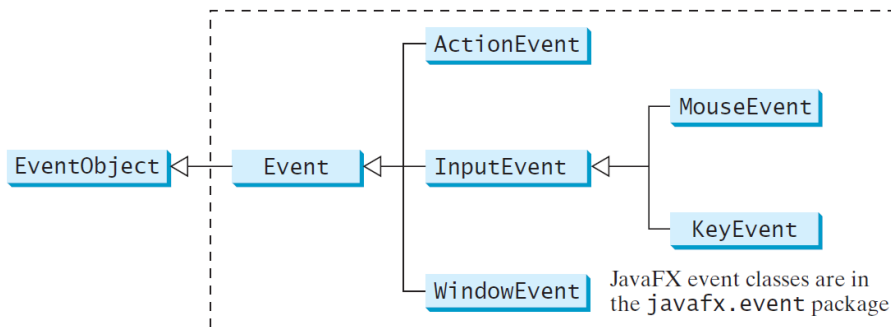
- ❑ An *event* can be defined as a type of signal to the program that something has happened.
- ❑ The event is generated by external user actions such as mouse movements, mouse clicks, or keystrokes.



Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

8

Event Classes



Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

9

Event Information

An event object contains whatever properties are pertinent to the event. You can identify the source object of the event using the `getSource()` instance method in the `EventObject` class. The subclasses of `EventObject` deal with special types of events, such as button actions, window events, mouse movements, and keystrokes. Table 16.1 lists external user actions, source objects, and event types generated.

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

10

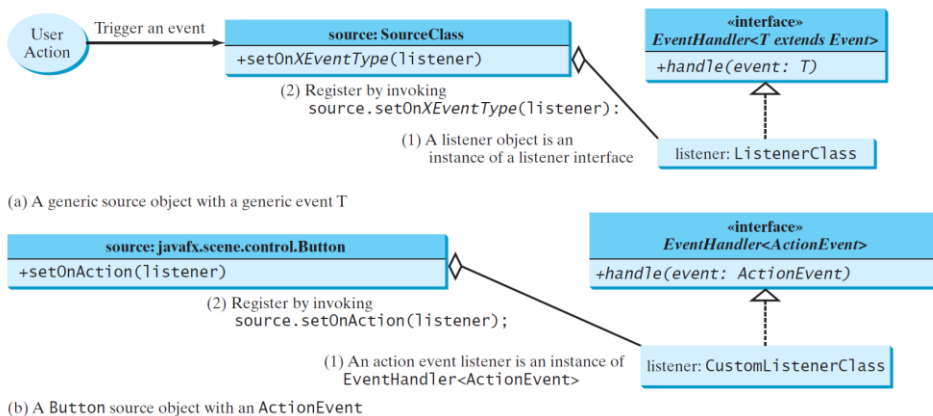
Selected User Actions and Handlers

User Action	Source Object	Event Type Fired	Event Registration Method
Click a button	Button	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Press Enter in a text field	TextField	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Check or uncheck	RadioButton	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Check or uncheck	CheckBox	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Select a new item	ComboBox	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Mouse pressed	Node, Scene	MouseEvent	setOnMousePressed(EventHandler<MouseEvent>)
Mouse released			setOnMouseReleased(EventHandler<MouseEvent>)
Mouse clicked			setOnMouseClicked(EventHandler<MouseEvent>)
Mouse entered			setOnMouseEntered(EventHandler<MouseEvent>)
Mouse exited			setOnMouseExited(EventHandler<MouseEvent>)
Mouse moved			setOnMouseMoved(EventHandler<MouseEvent>)
Mouse dragged			setOnMouseDragged(EventHandler<MouseEvent>)
Key pressed	Node, Scene	KeyEvent	setOnKeyPressed(EventHandler<KeyEvent>)
Key released			setOnKeyReleased(EventHandler<KeyEvent>)
Key typed			setOnKeyTyped(EventHandler<KeyEvent>)

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

11

The Delegation Model



Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

12

The Delegation Model: Example

```
Button btOK = new Button("OK");  
OKHandlerClass handler = new OKHandlerClass();  
btOK.setAction(handler);
```

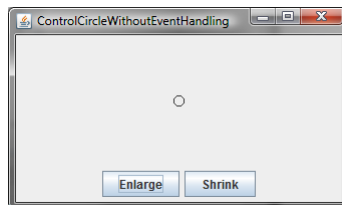


Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

13

Example: First Version for ControlCircle (no listeners)

Now let us consider to write a program that uses two buttons to control the size of a circle.



ControlCircleWithoutEventHandling

Run

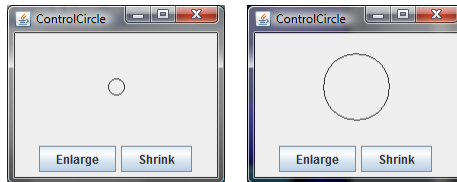


Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

14

Example: Second Version for ControlCircle (with listener for Enlarge)

Now let us consider to write a program that uses two buttons to control the size of a circle.



ControlCircle

Run

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

15

Inner Class Listeners

A listener class is designed specifically to create a listener object for a GUI component (e.g., a button). It will not be shared by other applications. So, it is appropriate to define the listener class inside the frame class as an inner class.

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

16

Inner Classes

Inner class: A class is a member of another class.

Advantages: In some applications, you can use an inner class to make programs simple.

An inner class can reference the data and methods defined in the outer class in which it nests, so you do not need to pass the reference of the outer class to the constructor of the inner class.

ShowInnerClass



Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

17

Inner Classes, cont.

```
public class Test {  
    ...  
  
    public class A {  
        ...  
    }  
}
```

(a)

```
public class Test {  
    ...  
  
    // Inner class  
    public class A {  
        ...  
    }  
}
```

(b)

```
// OuterClass.java: inner class demo  
public class OuterClass {  
    private int data;  
  
    /** A method in the outer class */  
    public void m() {  
        // Do something  
    }  
  
    // An inner class  
    class InnerClass {  
        /** A method in the inner class */  
        public void mi() {  
            // Directly reference data and method  
            // defined in its outer class  
            data++;  
            m();  
        }  
    }  
}
```

(c)

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

18

Inner Classes (cont.)

Inner classes can make programs simple and concise.

An inner class supports the work of its containing outer class and is compiled into a class named

OuterClassName\$InnerClassName.class.

For example, the inner class `InnerClass` in `OuterClass` is compiled into *OuterClass\$InnerClass.class*.



Inner Classes (cont.)

- ❑ An inner class can be declared public, protected, or private subject to the same visibility rules applied to a member of the class.
- ❑ An inner class can be declared static. A static inner class can be accessed using the outer class name. A static inner class cannot access nonstatic members of the outer class



Anonymous Inner Classes

- ❑ An anonymous inner class must always extend a superclass or implement an interface, but it cannot have an explicit extends or implements clause.
- ❑ An anonymous inner class must implement all the abstract methods in the superclass or in the interface.
- ❑ An anonymous inner class always uses the no-arg constructor from its superclass to create an instance. If an anonymous inner class implements an interface, the constructor is Object().
- ❑ An anonymous inner class is compiled into a class named OuterClassName\$n.class. For example, if the outer class Test has two anonymous inner classes, these two classes are compiled into Test\$1.class and Test\$2.class.



Anonymous Inner Classes (cont.)

Inner class listeners can be shortened using anonymous inner classes. An *anonymous inner class* is an inner class without a name. It combines declaring an inner class and creating an instance of the class in one step. An anonymous inner class is declared as follows:

```
new SuperClassName/InterfaceName() {  
    // Implement or override methods in superclass or interface  
    // Other methods if necessary  
}
```



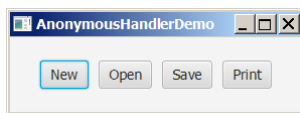
Anonymous Inner Classes (cont.)

```
public void start(Stage primaryStage) {  
    // Omitted  
  
    btEnlarge.setOnAction(  
        new EnlargeHandler();  
    );  
  
    class EnlargeHandler  
        implements EventHandler<ActionEvent> {  
        public void handle(ActionEvent e) {  
            circlePane.enlarge();  
        }  
    }  
}
```

(a) Inner class EnlargeListener

```
public void start(Stage primaryStage) {  
    // Omitted  
  
    btEnlarge.setOnAction(  
        new class EnlargeHandler  
            implements EventHandler<ActionEvent> {  
            public void handle(ActionEvent e) {  
                circlePane.enlarge();  
            }  
        }  
    );  
}
```

(b) Anonymous inner class



/AnonymousHandlerDemo

Run



Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

23

Simplifying Event Handling Using Lambda Expressions

Lambda expression is a new feature in Java 8. Lambda expressions can be viewed as an anonymous method with a concise syntax. For example, the following code in (a) can be greatly simplified using a lambda expression in (b) in three lines.

```
btEnlarge.setOnAction(  
    new EventHandler<ActionEvent>() {  
        @Override  
        public void handle(ActionEvent e) {  
            // Code for processing event e  
        }  
    }  
);
```

(a) Anonymous inner class event handler

```
btEnlarge.setOnAction(e -> {  
    // Code for processing event e  
});
```

(b) Lambda expression event handler

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

24

Basic Syntax for a Lambda Expression

The basic syntax for a lambda expression is either

`(type1 param1, type2 param2, ...) -> expression`

or

`(type1 param1, type2 param2, ...) -> { statements; }`

The data type for a parameter may be explicitly declared or implicitly inferred by the compiler. The parentheses can be omitted if there is only one parameter without an explicit data type.

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

25

Single Abstract Method Interface (SAM)

The statements in the lambda expression is all for that method. If it contains multiple methods, the compiler will not be able to compile the lambda expression. So, for the compiler to understand lambda expressions, the interface must contain exactly one abstract method. Such an interface is known as a *functional interface*, or a *Single Abstract Method (SAM)* interface.

AnonymousHandlerDemo

Run

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

26

Problem: Loan Calculator

LoanCalculator

Run



Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

27

The MouseEvent Class

`javafx.scene.input.MouseEvent`

```
+getButton(): MouseButton  
+getClickCount(): int  
+getX(): double  
+getY(): double  
+getSceneX(): double  
+getSceneY(): double  
+getScreenX(): double  
+getScreenY(): double  
+isAltDown(): boolean  
+isControlDown(): boolean  
+isMetaDown(): boolean  
+isShiftDown(): boolean
```

Indicates which mouse button has been clicked.
Returns the number of mouse clicks associated with this event.
Returns the *x*-coordinate of the mouse point in the event source node.
Returns the *y*-coordinate of the mouse point in the event source node.
Returns the *x*-coordinate of the mouse point in the scene.
Returns the *y*-coordinate of the mouse point in the scene.
Returns the *x*-coordinate of the mouse point in the screen.
Returns the *y*-coordinate of the mouse point in the screen.
Returns true if the **Alt** key is pressed on this event.
Returns true if the **Control** key is pressed on this event.
Returns true if the mouse **Meta** button is pressed on this event.
Returns true if the **Shift** key is pressed on this event.

MouseEventDemo

Run



Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

28

The KeyEvent Class

javafx.scene.input.KeyEvent

```
+getCharacter(): String  
+getCode(): KeyCode  
+getText(): String  
+isAltDown(): boolean  
+isControlDown(): boolean  
+isMetaDown(): boolean  
+isShiftDown(): boolean
```

Returns the character associated with the key in this event.
Returns the key code associated with the key in this event.
Returns a string describing the key code.
Returns true if the **Alt** key is pressed on this event.
Returns true if the **Control** key is pressed on this event.
Returns true if the mouse **Meta** button is pressed on this event.
Returns true if the **Shift** key is pressed on this event.

KeyEventDemo

Run

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

29

The KeyCode Constants

Constant	Description	Constant	Description
HOME	The Home key	CONTROL	The Control key
END	The End key	SHIFT	The Shift key
PAGE_UP	The Page Up key	BACK_SPACE	The Backspace key
PAGE_DOWN	The Page Down key	CAPS	The Caps Lock key
UP	The up-arrow key	NUM_LOCK	The Num Lock key
DOWN	The down-arrow key	ENTER	The Enter key
LEFT	The left-arrow key	UNDEFINED	The keyCode unknown
RIGHT	The right-arrow key	F1 to F12	The function keys from F1 to F12
ESCAPE	The Esc key	0 to 9	The number keys from 0 to 9
TAB	The Tab key	A to Z	The letter keys from A to Z

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

30

Example: Control Circle with Mouse and Key

ControlCircleWithMouseAndKey

Run



Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

31

Listeners for Observable Objects

You can add a listener to process a value change in an observable object.

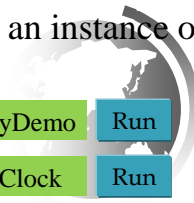
An instance of **Observable** is known as an *observable object*, which contains the **addListener(InvalidationListener listener)** method for adding a listener. Once the value is changed in the property, a listener is notified. The listener class should implement the **InvalidationListener** interface, which uses the **invalidated(Observable o)** method to handle the property value change. Every binding property is an instance of **Observable**.

ObservablePropertyDemo

Run

DisplayResizableClock

Run

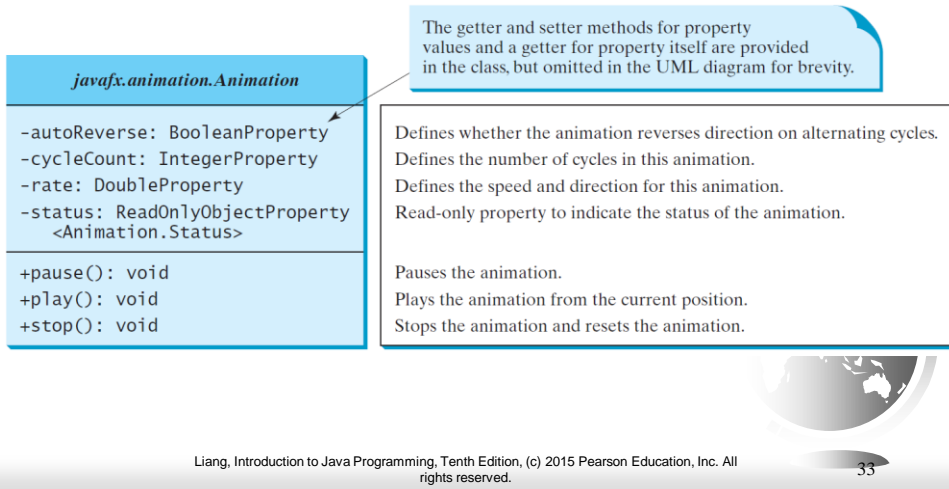


Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

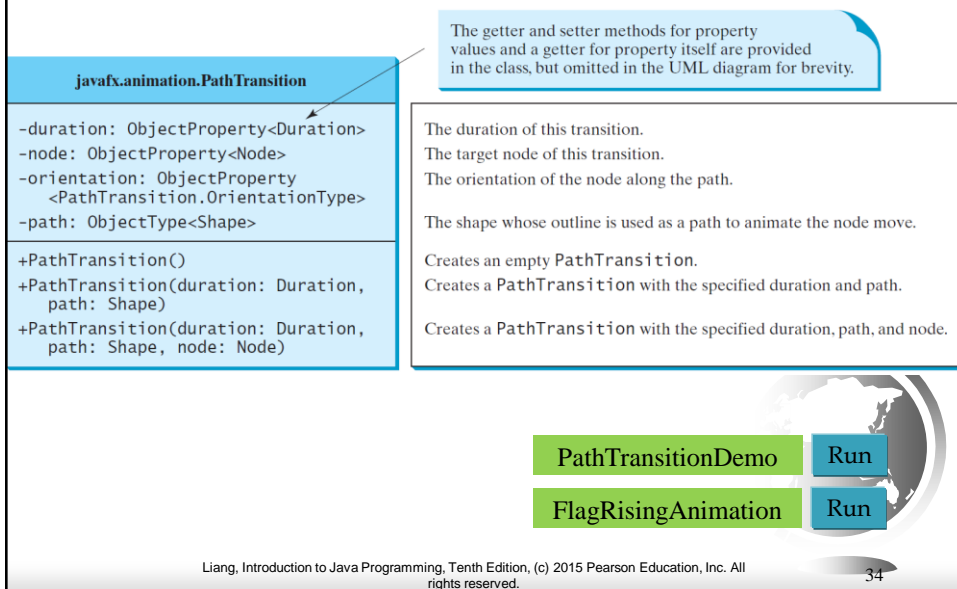
32

Animation

JavaFX provides the **Animation** class with the core functionality for all animations.



PathTransition



FadeTransition

The **FadeTransition** class animates the change of the opacity in 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.

FadeTransitionDemo

Run

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

35

Timeline

PathTransition and **FadeTransition** define specialized animations. The **Timeline** class can be used to program any animation using one or more **KeyFrames**. Each **KeyFrame** is executed sequentially at a specified time interval. **Timeline** inherits from **Animation**.

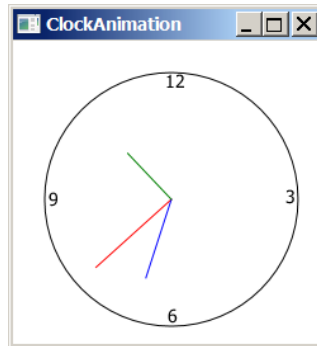
TimelineDemo

Run

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

36

Clock Animation



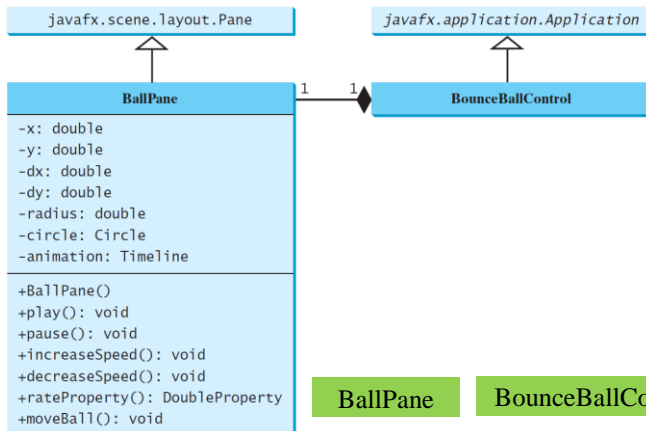
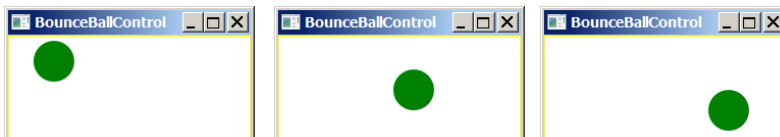
ClockAnimation

Run

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

37

Case Study: Bouncing Ball



BallPane

BounceBallControl

Run

Liang, Introduction to Java Programming, Tenth Edition, (c) 2015 Pearson Education, Inc. All rights reserved.

38