

Chapter 32 – Event-Driven Programming and Animations

Lia10, C15



Objectives

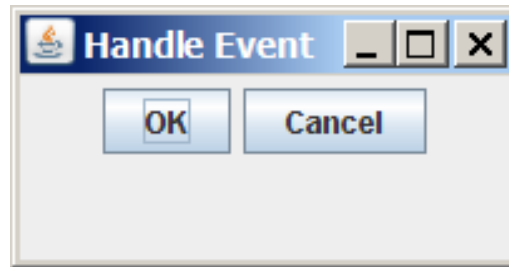
- To get a taste of event-driven programming (§15.1).
- To describe events, event sources, and event classes (§15.2).
- To define handler classes, register handler objects with the source object, and write the code to handle events (§15.3).
- To define handler classes using inner classes (§15.4).
- To define handler classes using anonymous inner classes (§15.5).
- To simplify event handling using lambda expressions (§15.6).
- To develop a GUI application for a loan calculator (§15.7).
- To write programs to deal with **MouseEvent**s (§15.8).
- To write programs to deal with **KeyEvent**s (§15.9).
- To use the **Animation**, **PathTransition**, **FadeTransition**, and **Timeline** classes to develop animations (§15.11).



Example: HandleEvent

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

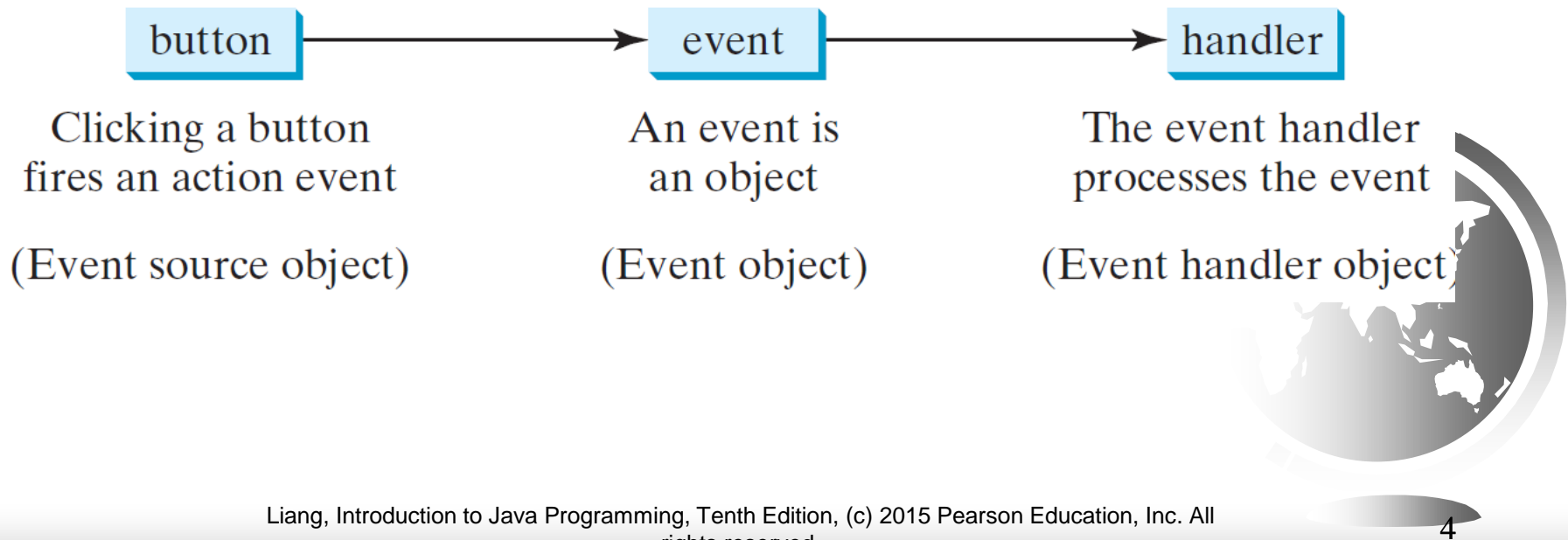
If the console window does not appear, end BlueJ and start it again.



Handling GUI Events

Source object (e.g., button)

Listener object contains a method for processing the event.



Trace Execution

```
public class HandleEvent extends Application {
```

```
    public void start(Stage primaryStage) {
```

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

```
        ...
```

```
        OKHandlerClass handler1 = new OKHandlerClass();
```

```
        btOK.setOnAction(handler1);
```

```
        CancelHandlerClass handler2 = new CancelHandlerClass();
```

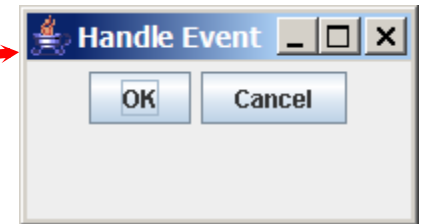
```
        btCancel.setOnAction(handler2);
```

```
        ...
```

```
        primaryStage.show(); // Display the stage
```

```
    }
```

```
}
```



```
class OKHandlerClass implements EventHandler<ActionEvent> {
```

```
    @Override
```

```
    public void handle(ActionEvent e) {
```

```
        System.out.println("OK button clicked");
```

```
    }
```

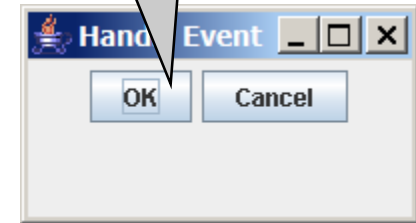
```
}
```



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



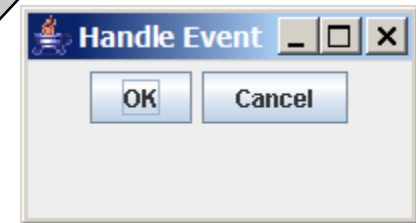
```
class OKHandlerClass implements EventHandler<ActionEvent> {  
    @Override  
    public void handle(ActionEvent e) {  
        System.out.println("OK button clicked");  
    }  
}
```



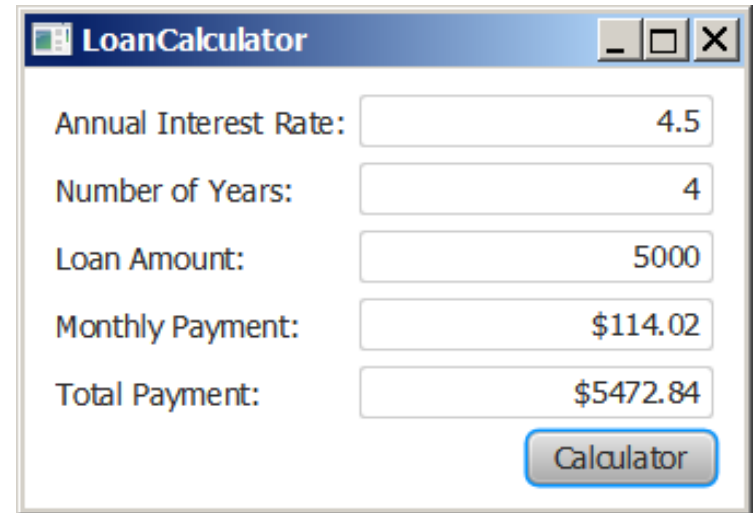
Trace Execution

```
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



Example: LoanCalculator



The screenshot shows a Java Swing window titled "LoanCalculator". It contains five input fields for loan parameters and their calculated values, along with a "Calculator" button.

Parameter	Value
Annual Interest Rate:	4.5
Number of Years:	4
Loan Amount:	5000
Monthly Payment:	\$114.02
Total Payment:	\$5472.84

Calculator

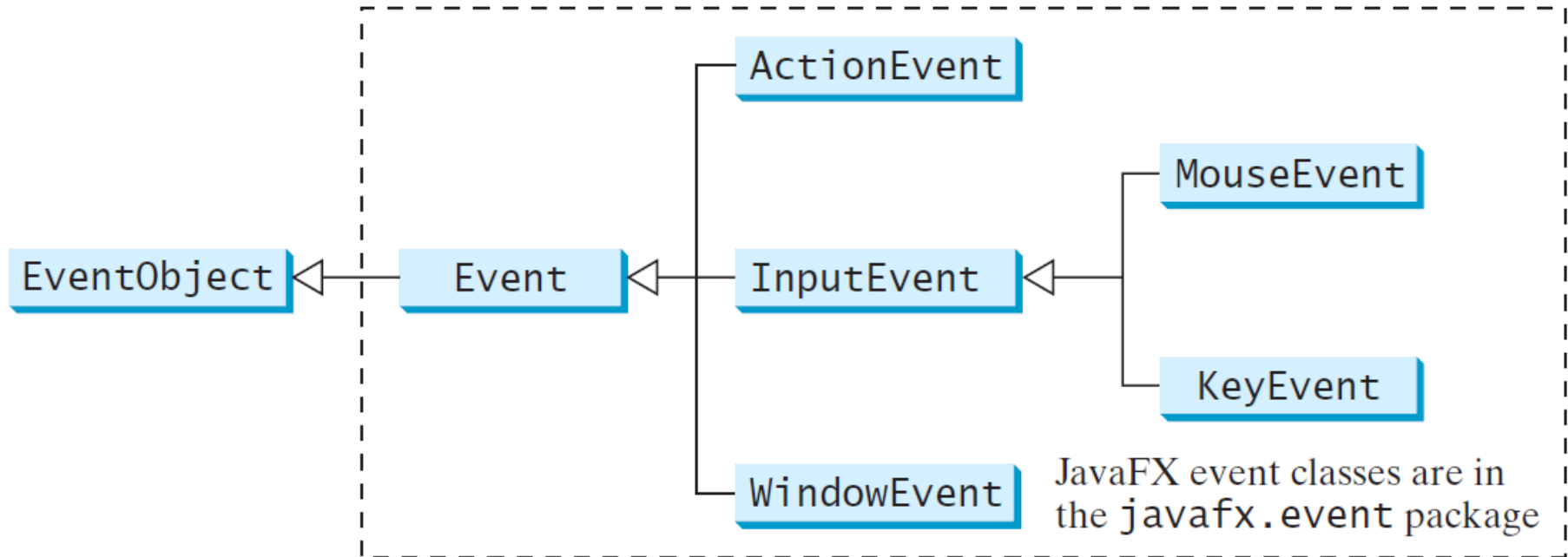


Events

- ❑ 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.



Event Classes



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. The following table lists external user actions, source objects, and event types generated.



Selected User Actions and Handlers

<i>User Action</i>	<i>Source Object</i>	<i>Event Type Fired</i>	<i>Event Registration Method</i>
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		KeyEvent	setOnKeyPressed(EventHandler<KeyEvent>)
Key released			setOnKeyReleased(EventHandler<KeyEvent>)
Key typed			setOnKeyTyped(EventHandler<KeyEvent>)

Listener Class

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



Writing an event with a handler class: **HandleEvent**

The following is about the “OK” button & event:

1. Write the handler class (**OKHandlerClass**), placing it outside the application, at the end of the same source file
2. Create and register the “OK” button with the handler.



Writing an event with a handler class: **HandleEvent** (cont.)

```
public class HandleEvent extends Application {
    @Override // Override the start method ...
    public void start(Stage primaryStage) {
        // Create buttons and register with handlers
        Button btOK = new Button("OK");
        btOK.setOnAction(new OKHandlerClass());
        ...
    }
}
...
// Handler for "OK" button
class OKHandlerClass implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        System.out.println("HandleEvent: OK button clicked");
    }
}
```

2

1



Writing an event with a handler as an inner class: **HandleEventWinnerClass**

The following is about the “OK” button & event:

1. Write the handler class (**OKHandlerClass**), placing it inside the application, at the beginning.
2. Create and register the “OK” button with the handler.



Writing an event with a handler as an inner class: **HandleEventWinnerClass** (cont.)

```
public class HandleEventWinnerClass extends Application {
    @Override // Override the start method
    public void start(Stage primaryStage) {
        // Handler for "OK" button
        class OKHandlerClass implements EventHandler<ActionEvent> {
            @Override
            public void handle(ActionEvent e) {
                System.out.println("HandleEventWinnerClass: OK button
1 clicked");
            }
        }
        ...
        // Create buttons and register with handlers
        Button btOK = new Button("OK");
        btOK.setOnAction(new OKHandlerClass());
        ...
    }
}
```

1

2



Writing an event with a handler as an anonymous inner class: **HandleEventWAnonInnerClass**

The following is about the “OK” button & event:

Write the handler class inside the **setOnAction** call, where the class does not have a name.

It combines declaring an inner class and creating an instance of the class in one step.

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

Writing an event with a handler as an anonymous inner class: **HandleEventWAnonInnerClass** (cont.)

```
public class HandleEventWAnonInnerClass extends Application {
    @Override // Override the start method in the Application class
    public void start(Stage primaryStage) {

        // Create buttons and register with handlers
        Button btOK = new Button("OK");
        btOK.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent e) { // Handler for "OK"
                System.out.println("HandleEventWAnonInnerClass: OK
button clicked");
            }
        });
        ...
    }
    ...
}
```



Writing an event with a lambda expression: **HandleEventWLambdaExpr**

The following is about the “OK” button & event:

Write the lambda expression inside the **setOnAction** call (i.e. more concise syntax of the anonymous class)

The syntax for a lambda expression is either

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

Writing an event with a lambda expression: **HandleEventWLambdaExpr** (cont.)

```
public class HandleEventWLambdaExpr extends Application {
    @Override // Override the start method in the Application class
    public void start(Stage primaryStage) {

        // Create buttons and register with handlers
        Button btOK = new Button("OK");
        btOK.setOnAction((ActionEvent e) -> { // Handler for "OK" with
lambda expression
            System.out.println("HandleEventWLambdaExpr: OK button
clicked");
        });
        ...
    }
    ...
}
```



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.



Example **MouseEventDemo**

- Drag the text with the mouse



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.



Example **KeyEventDemo**

- Type with a single key, i.e., not in combination with another key such as the shift key



The KeyCode Constants

<i>Constant</i>	<i>Description</i>	<i>Constant</i>	<i>Description</i>
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



Example

`ControlCircleWithMouseAndKey`

- Enlarge the circle with the “Enlarge” button or right-click or up-arrow key
- Shrink the circle with the “Shrink” button or left-click or down-arrow key



Animation

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

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.

PathTransition

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.



Example PathTransitionDemo



Example **FlagRisingAnimation**



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.

Example **FadeTransitionDemo**



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**.



Example **TimelineDemo**



Application **LoanCalculator**

- The user enters 3 values:
 - interest rate
 - number of years for the loan
 - dollar amount of the loan
- The program calculates and display 2 values:
 - monthly payment
 - total payment



THE END

