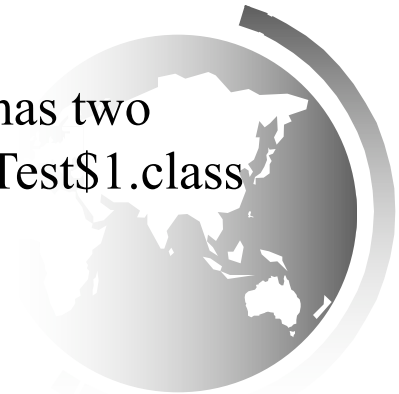


Event-Driven Programming (Anonymous Classes and Observable Objects)



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.
- ✦ 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.

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



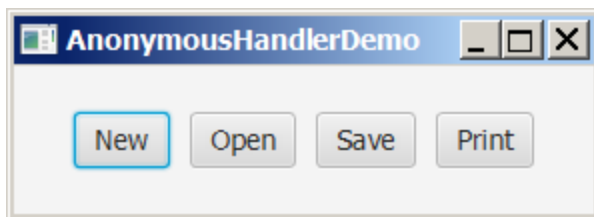
Anonymous Inner Classes (cont.)

```
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



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

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.



Single Abstract Method Interface (SAM)

- ◆ The statements in the lambda expression is all for that method.
- ◆ If interface contains multiple methods, the compiler will not be able to compile the lambda expression. So, 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



Problem: Loan Calculator

LoanCalculator



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

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



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: Control Circle with Mouse and Key

ControlCircleWithMouseAndKey



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

DisplayResizableClock