Java GUI Programming
Part 2

Chapter 12 and 22

P. Deitel , H. Deitel - Java How To Program, 10th Edition

Edited by Ehsan Edalat

12.6 Introduction to Event Handling with Nested Classes

- GUIs are event driven.
- When the user interacts with a GUI component, the interaction—known as an event—drives the program to perform a task.
- The code that performs a task in response to an event is called an event handler, and the process of responding to events is known as event handling.

12.6 Introduction to Event Handling with Nested Classes (cont.)

- Before an application can respond to an event for a particular GUI component, you must perform several coding steps:
  - Create a class that represents the event handler.
  - Implement an appropriate interface, known as an event-listener interface, in the class from *Step 1.*
  - Indicate that an object of the class from Steps 1 and 2 should be notified when the event occurs. This is known as registering the event handler.

12.6 Introduction to Event Handling with Nested Classes (cont.)

- All the classes discussed so far were so-called top-level classes—that is, they were not declared inside another class.
- Java allows you to declare classes inside other classes—these are called nested classes.
  - Can be `static` or non-`static`.
  - Non-`static` nested classes are called inner classes and are frequently used to implement event handlers.

12.6 Introduction to Event Handling with Nested Classes (cont.)

- Before an object of an inner class can be created, there must first be an object of the top-level class that contains the inner class.
- This is required because an inner-class object implicitly has a reference to an object of its top-level class.
- There is also a special relationship between these objects—the inner-class object is allowed to directly access all the variables and methods of the outer class.

12.6 Introduction to Event Handling with Nested Classes (cont.)

- Inner classes can be declared `public`, `protected` or `private`.
- Since event handlers tend to be specific to the application in which they are defined, they are often implemented as `private` inner classes.

12.6 Introduction to Event Handling with Nested Classes (cont.)

- GUI components can generate many events in response to user interactions.
- Each event is represented by a class and can be processed only by the appropriate type of event handler.
- Normally, a component’s supported events are described in the Java API documentation for that component’s class and its superclasses.

12.7 Common GUI Event Types and Listener Interfaces

- Figure 12.11 illustrates a hierarchy containing many event classes from the package java.awt.event.
- Used with both AWT and Swing components.
- Additional event types that are specific to Swing GUI components are declared in package javax.swing.event.

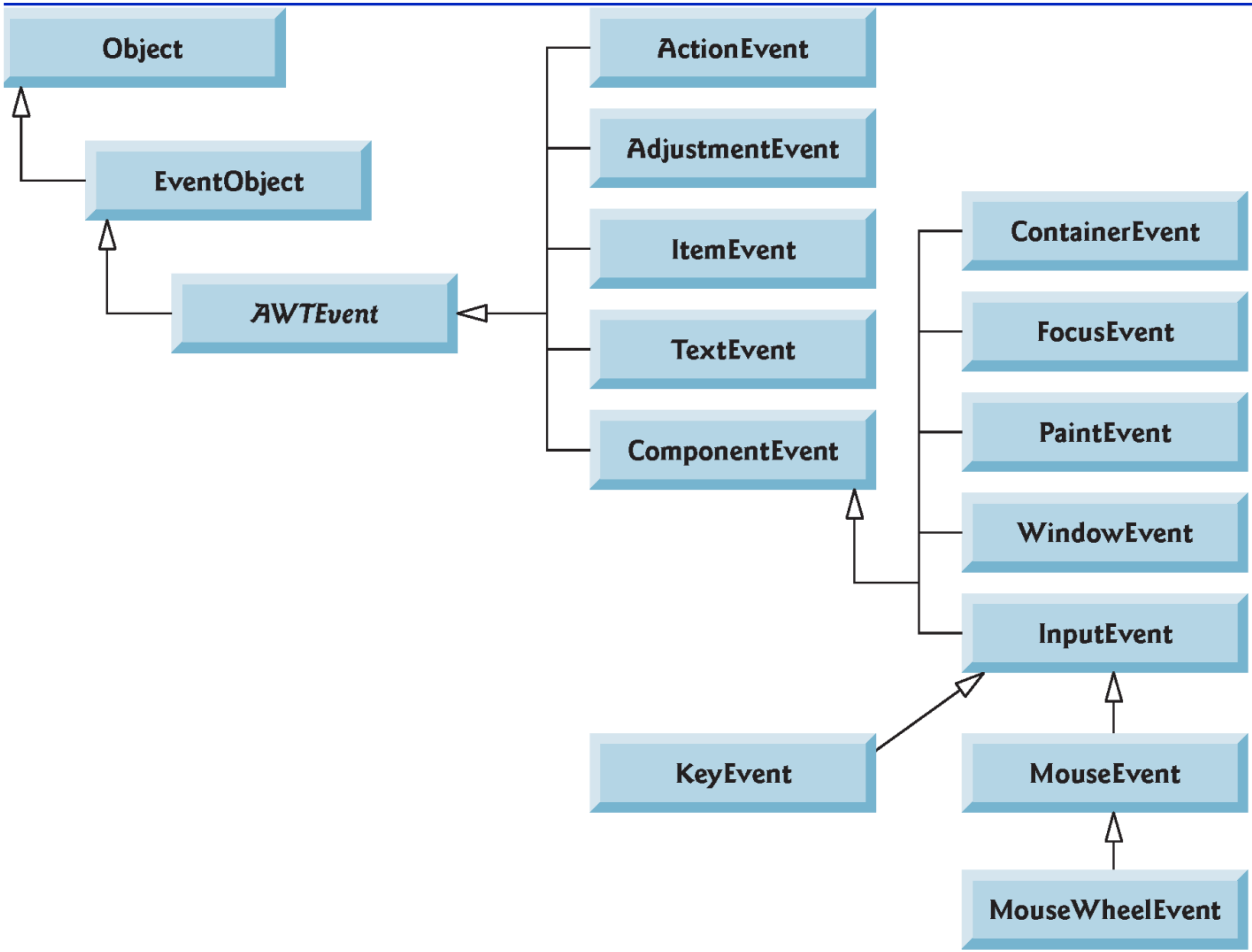


Fig. 12.11 | Some event classes of package `java.awt.event`.

12.7 Common GUI Event Types and Listener Interfaces (cont.)

- Delegation event model—an event's processing is delegated to an object (the event listener) in the application.
- For each event-object type, there is typically a corresponding event-listener interface.
- Many event-listener types are common to both Swing and AWT components.
  - Such types are declared in package `java.awt.event`, and some of them are shown in Fig. 12.12.
- Additional event-listener types that are specific to Swing components are declared in package `javax.swing.event`.

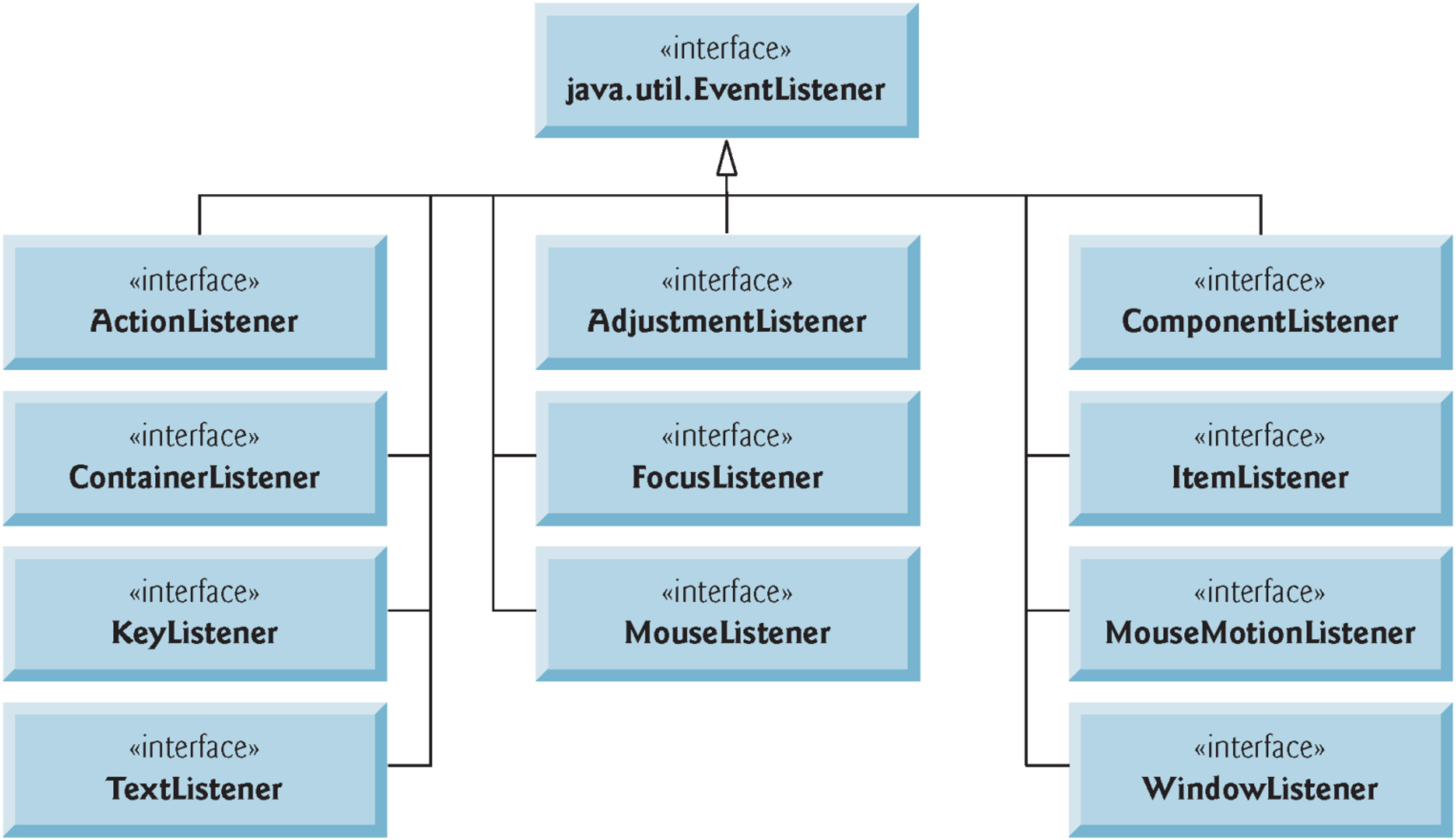
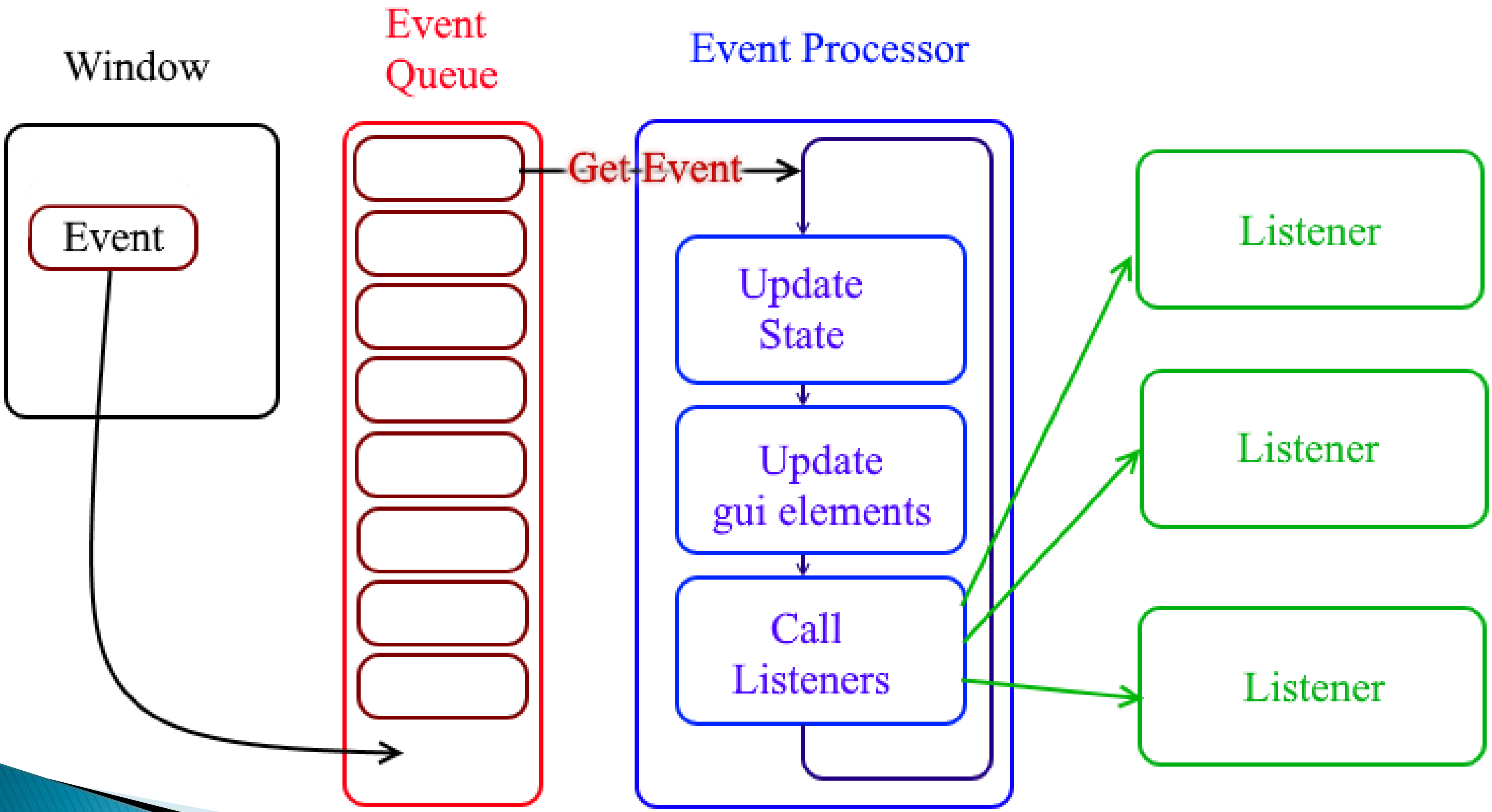


Fig. 12.12 | Some common event-listener interfaces of package `java.awt.event`.

12.7 Common GUI Event Types and Listener Interfaces (cont.)

- Each event-listener interface specifies one or more event-handling methods that must be declared in the class that implements the interface.
- When an event occurs, the GUI component with which the user interacted notifies its registered listeners by calling each listener’s appropriate event-handling method.

How Event Handling Works



12.8 How Event Handling Works

- How the event-handling mechanism works:
- Every `JComponent` has a field `listenerList` that refers to an EventListenerList (package `javax.swing.event`).
- Maintains references to *registered listeners* in the `listenerList`.
- When a listener is registered, a new entry is placed in the component's `listenerList`.
- Every entry also includes the listener's type.

12.8 How Event Handling Works (cont.)

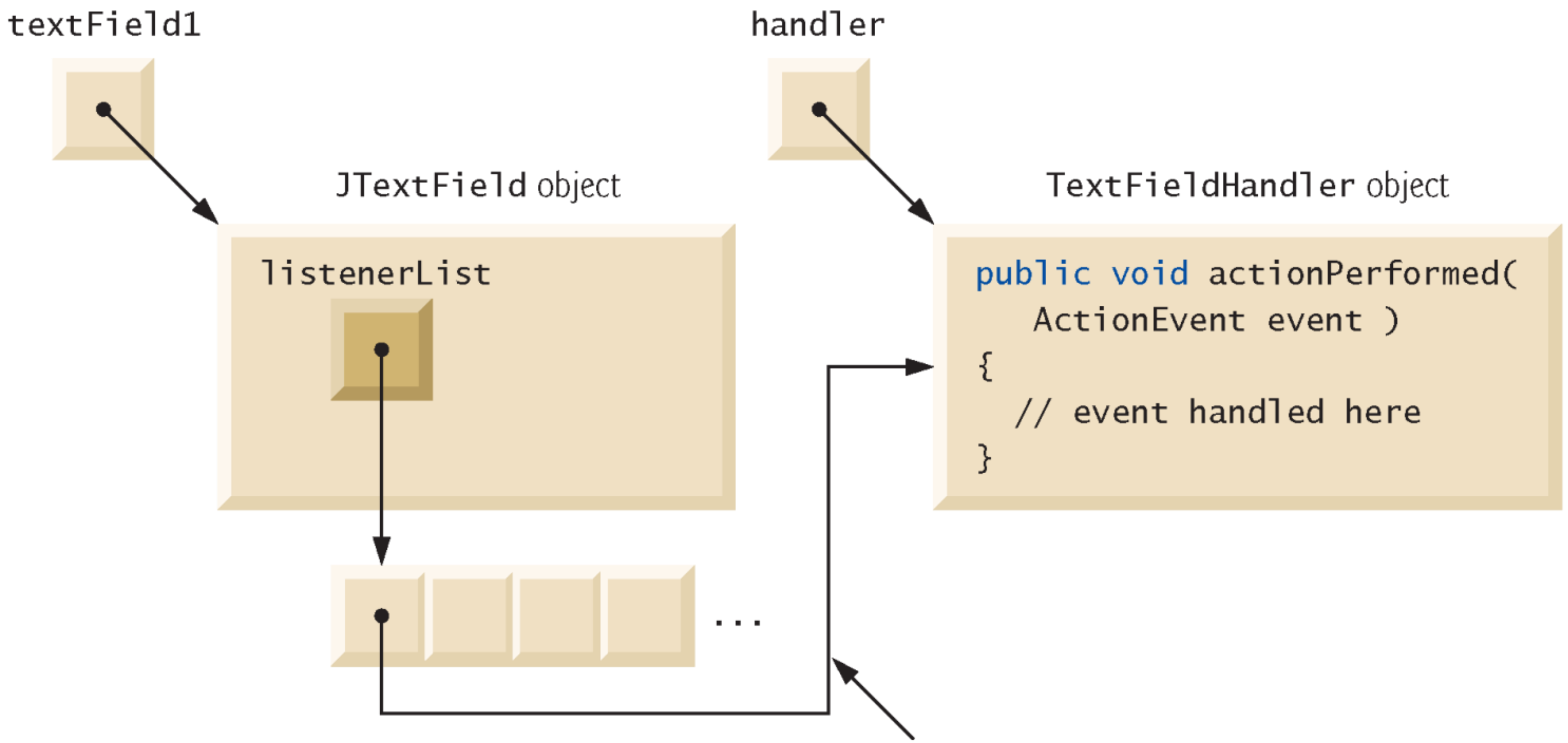
- How does the GUI component know to call `actionPerformed` rather than another method?
  - Every GUI component supports several event types, including mouse events, key events and others.
  - When an event occurs, the event is dispatched only to the event listeners of the appropriate type.
  - Dispatching is simply the process by which the GUI component calls an event-handling method on each of its listeners that are registered for the event type that occurred.

12.8 How Event Handling Works (cont.)

- Each *event type* has one or more corresponding event-listener interfaces.
  - ActionEvents are handled by ActionListeners
  - MouseEvents are handled by MouseListeners and MouseMotionListeners
  - KeyEvents are handled by KeyListeners
- When an event occurs, the GUI component receives (from the JVM) a unique event ID specifying the event type.
  - The component uses the event ID to decide the listener type to which the event should be dispatched and to decide which method to call on each listener object.

12.8 How Event Handling Works (cont.)

- For an `ActionEvent`, the event is dispatched to every registered `ActionListener`'s `actionPerformed` method.
- For a `Mouse-Event`, the event is dispatched to every registered `MouseListener` or `MouseMotionListener`, depending on the mouse event that occurs.
  - The `MouseEvent`'s event ID determines which of the several mouse event-handling methods are called.

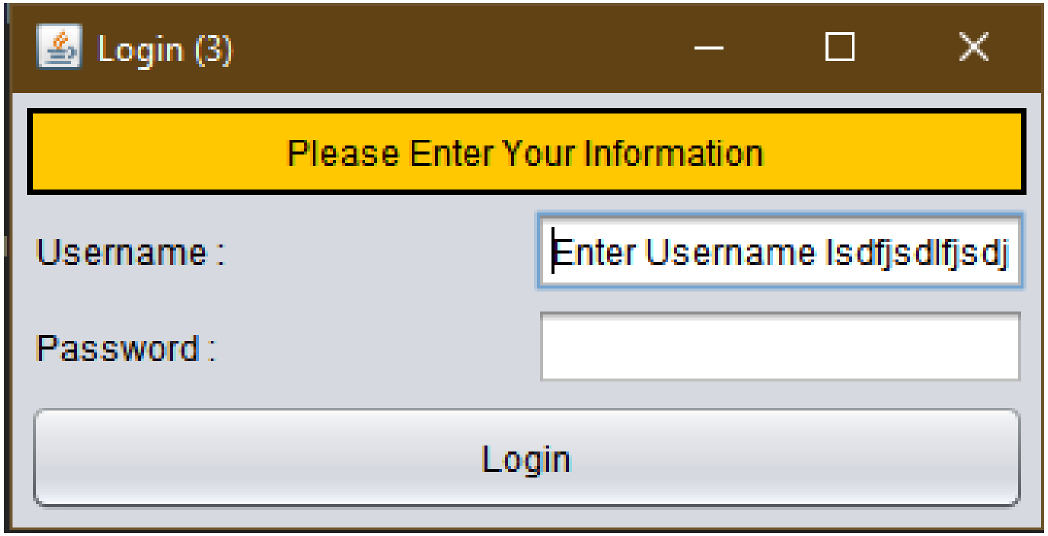


```
public void actionPerformed(
   ActionEvent event )
{
  // event handled here
}
```

This reference is created by the statement
`textField1.addActionListener( handler );`

Fig. 12.13 | Event registration for `JTextField textField1`.

LoginFrame-ActionListener



Login (3)

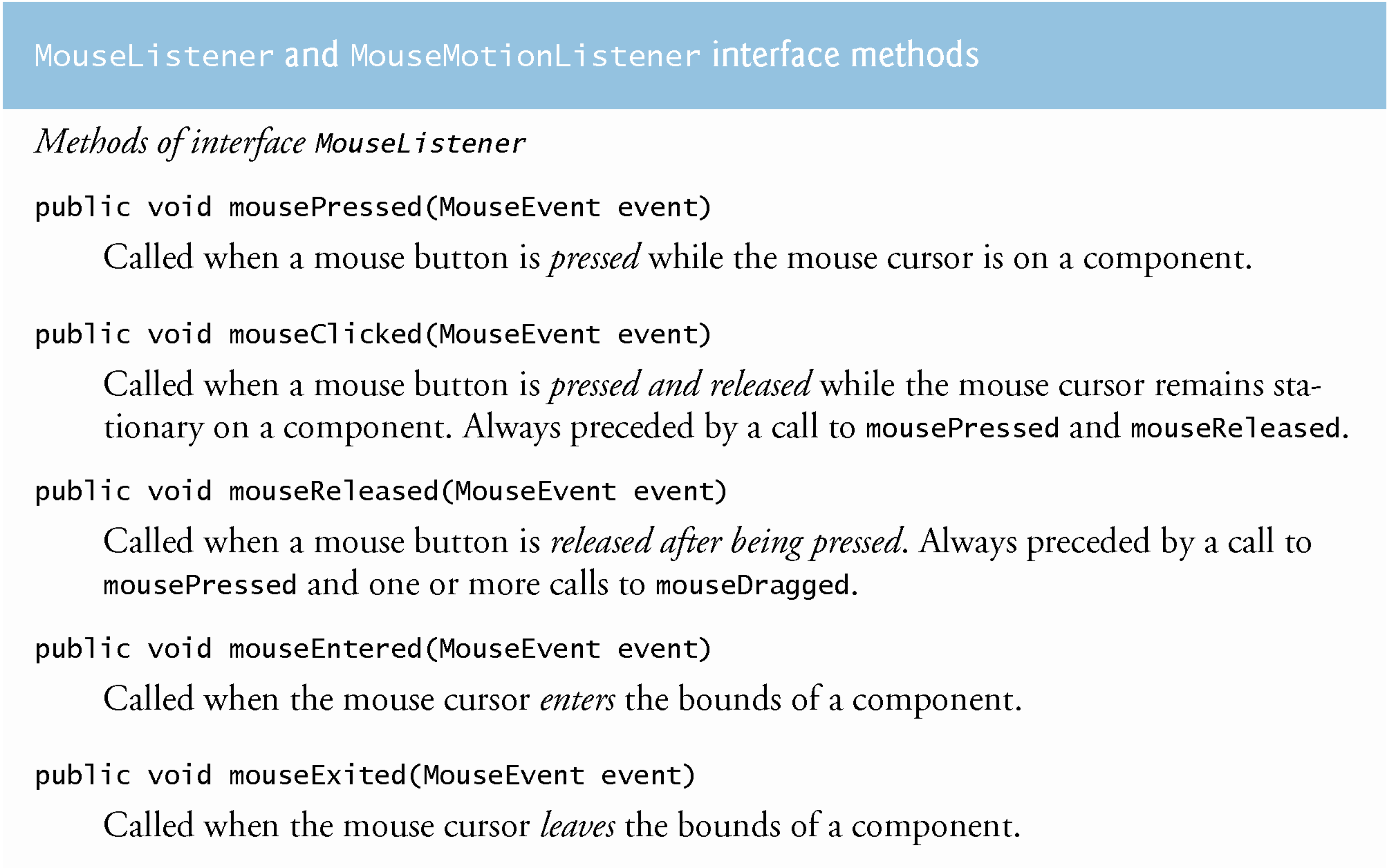
Please Enter Your Information

Username : Enter Username lsdfjsdlfjsdj

Password :

12.14 Mouse Event Handling

- MouseListener and MouseMotionListener event-listener interfaces for handling mouse events.
    - Any GUI component
- Package `javax.swing.event` contains interface MouseInputListener, which extends interfaces `MouseListener` and `MouseMotionListener` to create a single interface containing all the methods.
- `MouseListener` and `MouseMotionListener` methods are called when the mouse interacts with a `Component` if appropriate event-listener objects are registered for that `Component`.



`MouseListener` and `MouseMotionListener` interface methods

Methods of interface `MouseListener`

`public void mousePressed(MouseEvent event)`

Called when a mouse button is *pressed* while the mouse cursor is on a component.

`public void mouseClicked(MouseEvent event)`

Called when a mouse button is *pressed and released* while the mouse cursor remains stationary on a component. Always preceded by a call to `mousePressed` and `mouseReleased`.

`public void mouseReleased(MouseEvent event)`

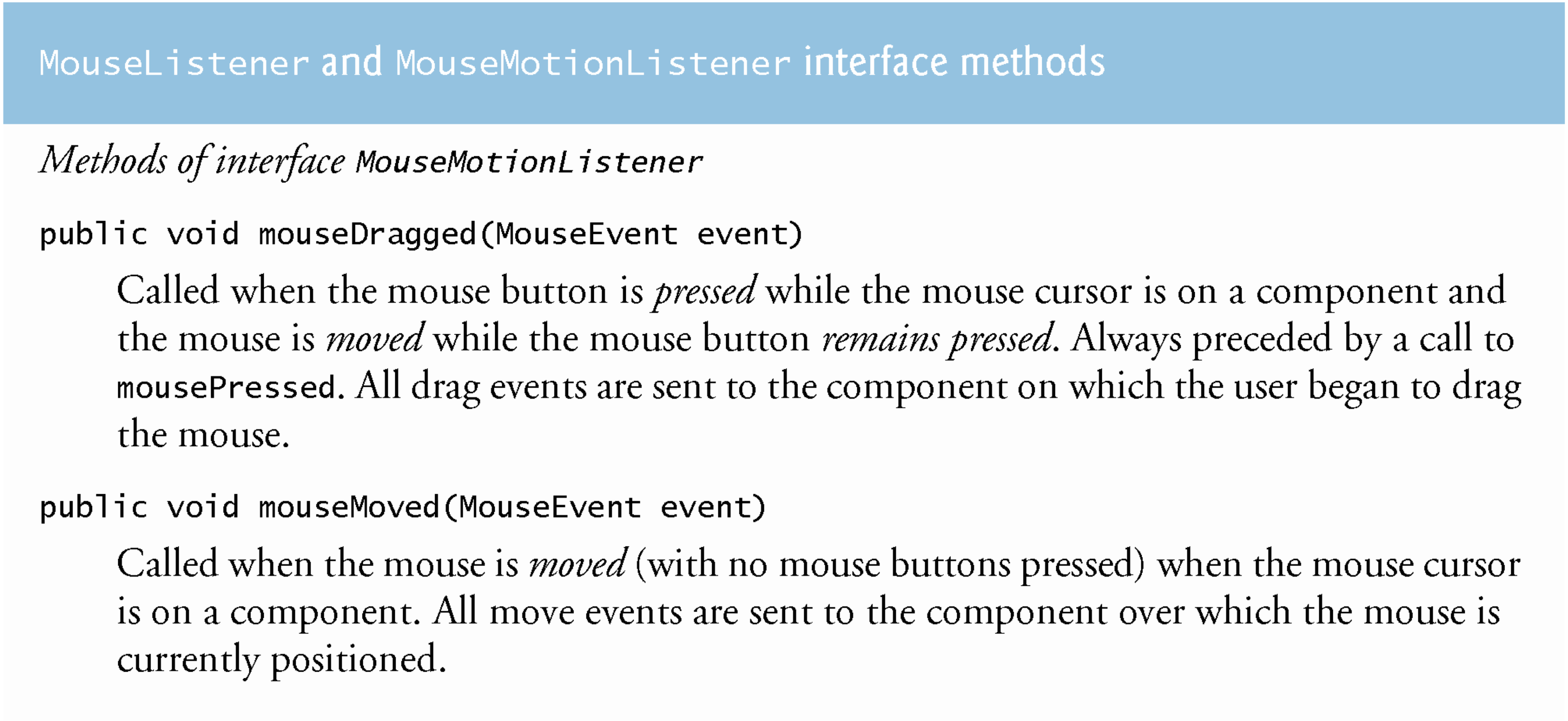
Called when a mouse button is *released after being pressed*. Always preceded by a call to `mousePressed` and one or more calls to `mouseDragged`.

`public void mouseEntered(MouseEvent event)`

Called when the mouse cursor *enters* the bounds of a component.

`public void mouseExited(MouseEvent event)`

Called when the mouse cursor *leaves* the bounds of a component.



`MouseListener` and `MouseMotionListener` interface methods

Methods of interface `MouseMotionListener`

`public void mouseDragged(MouseEvent event)`

Called when the mouse button is *pressed* while the mouse cursor is on a component and the mouse is *moved* while the mouse button *remains pressed*. Always preceded by a call to `mousePressed`. All drag events are sent to the component on which the user began to drag the mouse.

`public void mouseMoved(MouseEvent event)`

Called when the mouse is *moved* (with no mouse buttons pressed) when the mouse cursor is on a component. All move events are sent to the component over which the mouse is currently positioned.

Fig. 12.27 | `MouseListener` and `MouseMotionListener` interface methods. (Part 2 of 2.)

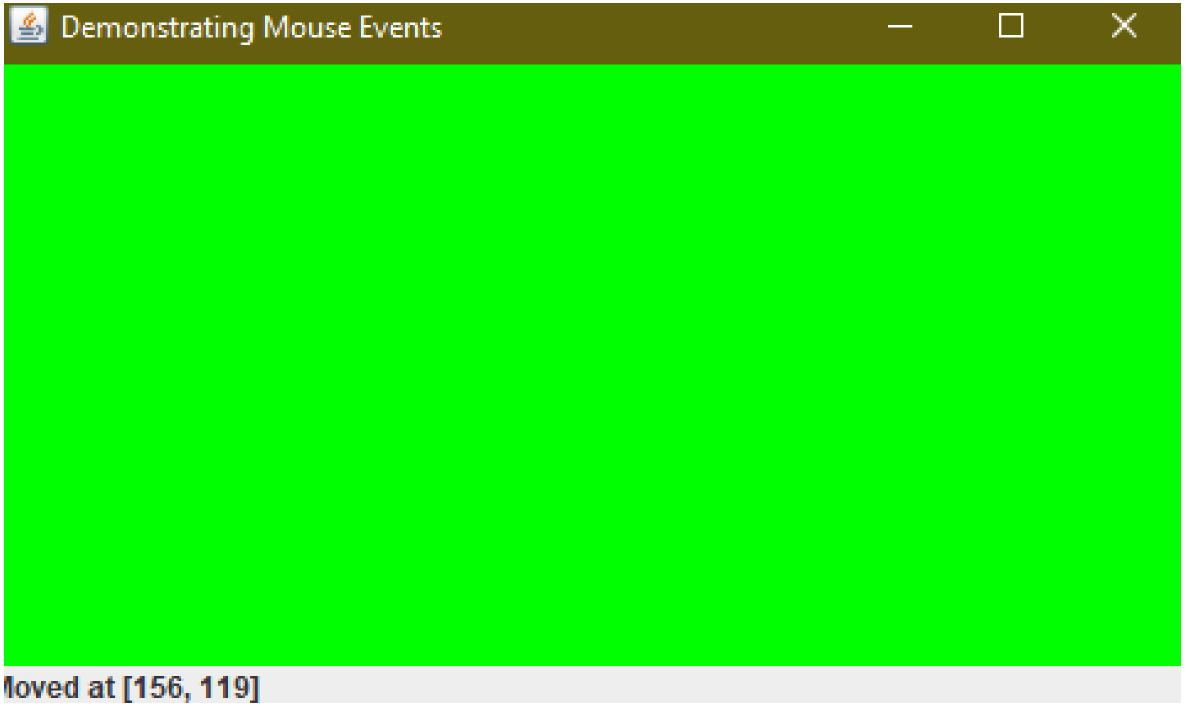
12.14 Mouse Event Handling (cont.)

- Each mouse event-handling method receives a MouseEvent object that contains information about the mouse event that occurred, including the *x*- and y-coordinates of the location where the event occurred.
- Coordinates are measured from the upper-left corner of the GUI component on which the event occurred.
- The x-coordinates start at 0 and increase from left to right. The *y*-coordinates start at 0 and increase from top to bottom.
- The methods and constants of class InputEvent (`MouseEvent`'s superclass) enable you to determine which mouse button the user clicked.

12.14 Mouse Event Handling (cont.)

- Interface MouseWheelListener enables applications to respond to the rotation of a mouse wheel.
- Method mouseWheelMoved receives a MouseWheelEvent as its argument.
- Class `MouseWheelEvent` (a subclass of `MouseEvent`) contains methods that enable the event handler to obtain information about the amount of wheel rotation.

12.14 Mouse Event Handling (cont.)

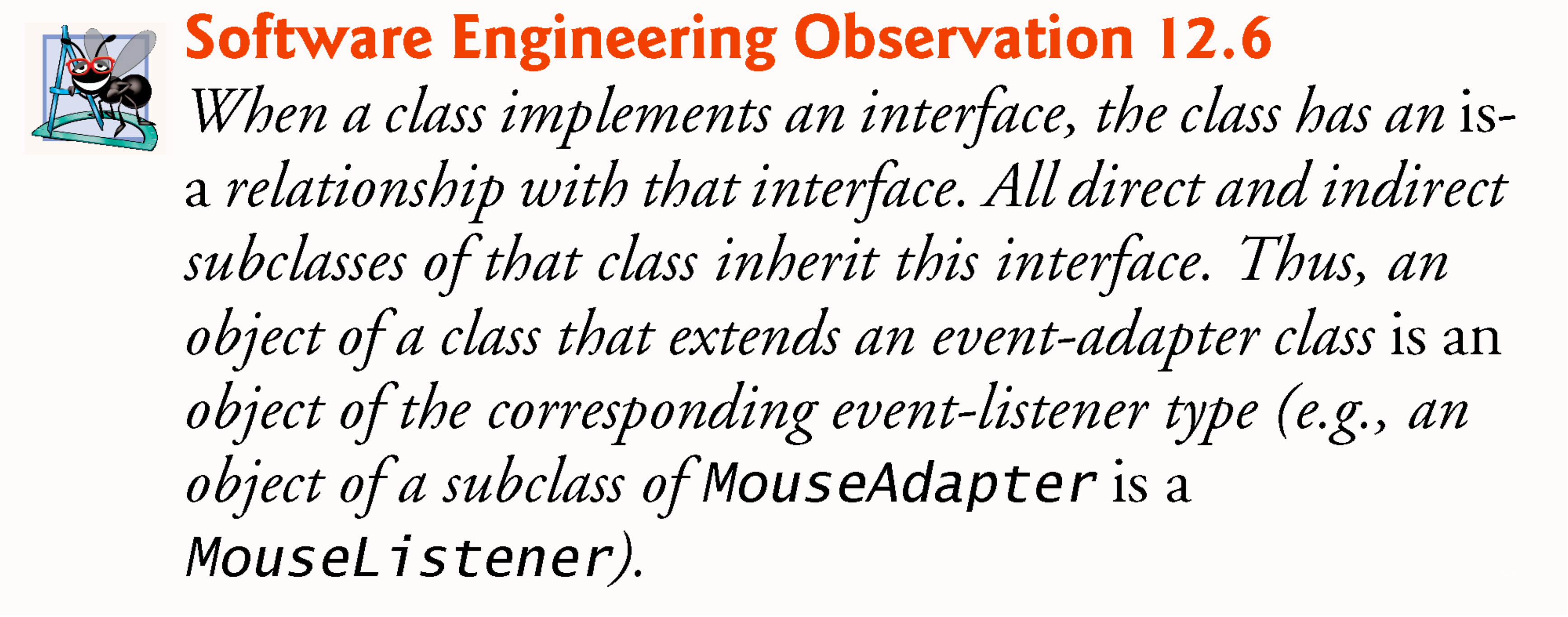


```
loved at [156, 119]
```

12.15 Adapter Classes

- Many event-listener interfaces contain multiple methods.
- An adapter class implements an interface and provides a default implementation (with an empty method body) of each method in the interface.
- You extend an adapter class to inherit the default implementation of every method and override only the method(s) you need for event handling.

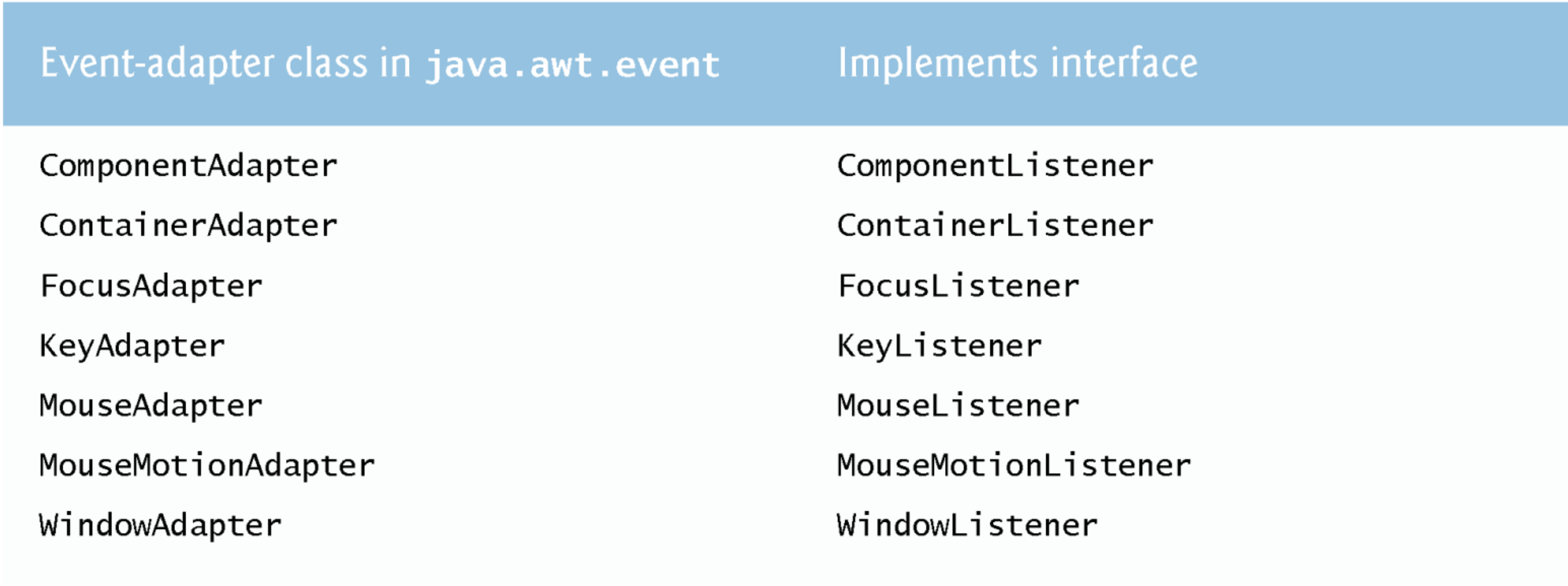
12.15 Adapter Classes (cont.)



Software Engineering Observation 12.6

When a class implements an interface, the class has an is-a *relationship with that interface. All direct and indirect subclasses of that class inherit this interface. Thus, an object of a class that extends an event-adapter class* is an *object of the corresponding event-listener type (e.g., an object of a subclass of* `MouseAdapter` is a `MouseListener`*).*

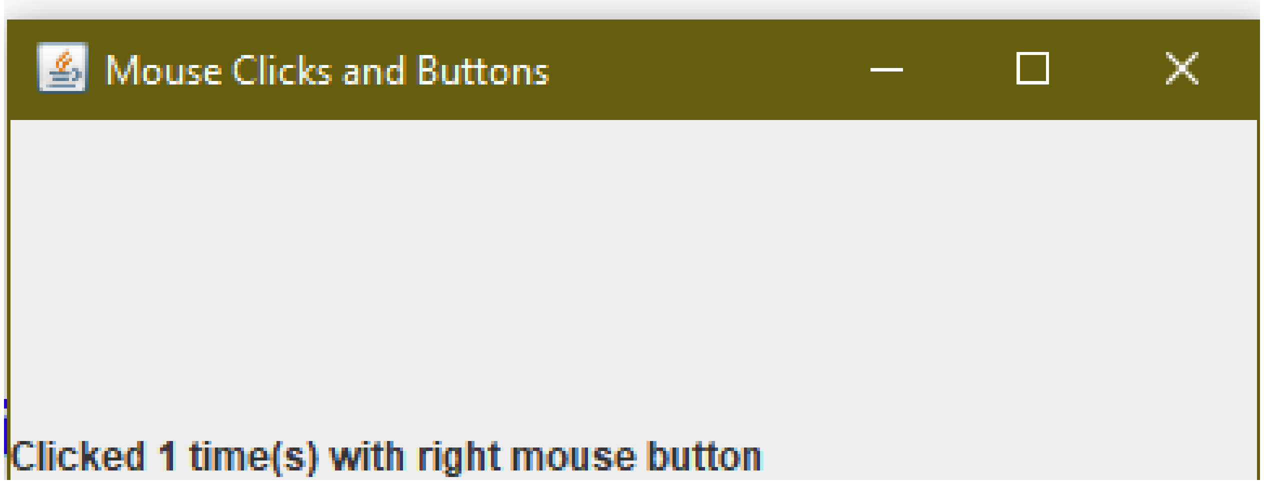
12.15 Adapter Classes (cont.)



Event-adapter class in `java.awt.event`	Implements interface
`ComponentAdapter`	`ComponentListener`
`ContainerAdapter`	`ContainerListener`
`FocusAdapter`	`FocusListener`
`KeyAdapter`	`KeyListener`
`MouseAdapter`	`MouseListener`
`MouseMotionAdapter`	`MouseMotionListener`
`WindowAdapter`	`WindowListener`

Fig. 12.30 | Event-adapter classes and the interfaces they implement.

12.15 Adapter Classes (cont.)



Clicked 1 time(s) with right mouse button

12.17 Key Event Handling

- `KeyListener` interface for handling key events.
- Key events are generated when keys on the keyboard are pressed and released.
- A `KeyListener` must define methods keyPressed, keyReleased and keyTyped
  - each receives a `KeyEvent` as its argument
- Class `KeyEvent` is a subclass of `InputEvent`.
- Method `keyPressed` is called in response to pressing any key.
- Method `keyTyped` is called in response to pressing any key that is not an action key.
- Method `keyReleased` is called when the key is released after any `keyPressed` or `keyTyped` event.

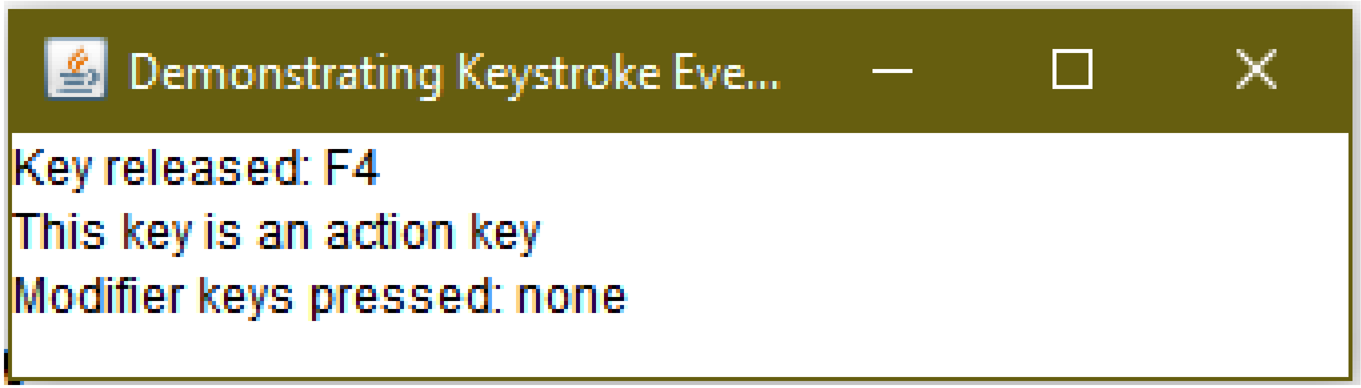
12.17 Key Event Handling (cont.)

- Registers key event handlers with method addKeyListener from class `Component`.
- `KeyEvent` method getKeyCode gets the virtual key code of the pressed key.
- `KeyEvent` contains virtual key-code constants that represents every key on the keyboard.
- Value returned by `getKeyCode` can be passed to `static KeyEvent` method getKeyText to get a string containing the name of the key that was pressed.
- `KeyEvent` method getKeyChar (which returns a `char`) gets the Unicode value of the character typed.
- `KeyEvent` method isActionKey determines whether the key in the event was an action key.

12.17 Key Event Handling (cont.)

- Method getModifiers determines whether any modifier keys (such as *Shift*, *Alt* and *Ctrl*) were pressed when the key event occurred.
  - Result can be passed to `static` `KeyEvent` method getKeyModifiersText to get a string containing the names of the pressed modifier keys.
- `InputEvent` methods isAltDown, isControlDown, isMetaDown and isShiftDown each return a `boolean` indicating whether the particular key was pressed during the key event.

12.17 Key Event Handling (cont.)



```
Key released: F4
This key is an action key
Modifier keys pressed: none
```

Separating Logics from UI

- You should have some classes for processing and maintaining data. (logic classes)
- Also, you should have some classes for adjusting and representing GUI. (UI classes)
- In event handling methods you should just call methods of logic classes.
- Best practice for this situation is MVC design pattern.
- Follow this link for more information:
- https://medium.com/@ssaurel/learn-to-make-a-mvc-application-with-swing-and-java-8-3cd24cf7cb10