Computer Graphics through Game Programming Event Handling

Introduction to Event Handling

- 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 overall process of responding to events is known as event handling.

(C) 2010 Pearson Education, Inc.
All rights reserved.

Introduction to Event Handling (cont.)

- `JTextFields` and `JPasswordFields` (package `javax.swing`).
- `JTextField` extends class `JTextComponent` (package `javax.swing.text`), which provides many features common to Swing's text-based components.
- Class `JPasswordField` extends `JTextField` and adds methods that are specific to processing passwords.
- `JPasswordField` shows that characters are being typed as the user enters them, but hides the actual characters with an echo character.

(C) 2010 Pearson Education, Inc.
All rights reserved.

```
1  // Fig. 14.9: TextFieldFrame.java
2  // Demonstrating the JTextField class.
3  import java.awt.FlowLayout;
4  import java.awt.event.ActionListener;
5  import java.awt.event.ActionEvent;
6  import javax.swing.JFrame;
7  import javax.swing.JTextField;
8  import javax.swing.JPasswordField;
9  import javax.swing.JOptionPane;
10
11 public class TextFieldFrame extends JFrame
12 {
13    private JTextField textField1; // text field with set size
14    private JTextField textField2; // text field constructed with text
15    private JTextField textField3; // text field with text and size
16    private JPasswordField passwordField; // password field with text
17
18    // TextFieldFrame constructor adds JTextFields to JFrame
19    public TextFieldFrame()
20    {
21       super( "Testing JTextField and JPasswordField" );
22       setLayout( new FlowLayout() ); // set frame layout
23
```

Fig. 14.9 | JTextFields and JPasswordFields. (Part 1 of 4.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

```
24          // construct textfield with 10 columns
25          textField1 = new JTextField( 10 );
26          add( textField1 ); // add textField1 to JFrame
27
28          // construct textfield with default text
29          textField2 = new JTextField( "Enter text here" );
30          add( textField2 ); // add textField2 to JFrame
31
32          // construct textfield with default text and 21 columns
33          textField3 = new JTextField( "Uneditable text field", 21 );
34          textField3.setEditable( false ); // disable editing
35          add( textField3 ); // add textField3 to JFrame
36
37          // construct passwordfield with default text
38          passwordField = new JPasswordField( "Hidden text" );
39          add( passwordField ); // add passwordField to JFrame
40
41          // register event handlers
42          TextFieldHandler handler = new TextFieldHandler();
43          textField1.addActionListener( handler );
44          textField2.addActionListener( handler );
45          textField3.addActionListener( handler );
46          passwordField.addActionListener( handler );
47       } // end TextFieldFrame constructor
```

Width of the `JTextField` is based on the component's current font unless a layout manager overrides that size.

Width of the `JTextField` is based on the default text unless a layout manager overrides that size.

Width based on second argument unless a layout manager overrides that size.

Text in this component will be hidden by asterisks (*) by default.

`TextFieldHandler` inner class implements `ActionListener` interface, so it can respond to `JTextField` events. Lines 43–46 register the object `handler` to respond to each component's events.

Fig. 14.9 | `JTextFields` and `JPasswordFields`. (Part 2 of 4.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

```
48
49    // private inner class for event handling
50    private class TextFieldHandler implements ActionListener
51    {
52       // process text field events
53       public void actionPerformed( ActionEvent event )
54       {
55          String string = ""; // declare string to display
56
57          // user pressed Enter in JTextField textField1
58          if ( event.getSource() == textField1 )
59             string = String.format( "textField1: %s",
60                event.getActionCommand() );
61
62          // user pressed Enter in JTextField textField2
63          else if ( event.getSource() == textField2 )
64             string = String.format( "textField2: %s",
65                event.getActionCommand() );
66
67          // user pressed Enter in JTextField textField3
68          else if ( event.getSource() == textField3 )
69             string = String.format( "textField3: %s",
70                event.getActionCommand() );
71
```

A `TextFieldHandler` is an `ActionListener`.

Called when the user presses *Enter* in a `JTextField` or `JPasswordField`.

`getSource` specifies which component the user interacted with

Obtains the text the user typed in the textfield.

Fig. 14.9 | `JTextFields` and `JPasswordFields`. (Part 3 of 4.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

```
72             // user pressed Enter in JTextField passwordField
73             else if ( event.getSource() == passwordField )
74                string = String.format( "passwordField: %s",
75                   event.getActionCommand() );
76
77             // display JTextField content
78             JOptionPane.showMessageDialog( null, string );
79          } // end method actionPerformed
80       } // end private inner class TextFieldHandler
81    } // end class TextFieldFrame
```

Fig. 14.9 | JTextFields and JPasswordFields. (Part 4 of 4.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

```java
1  // Fig. 14.10: TextFieldTest.java
2  // Testing TextFieldFrame.
3  import javax.swing.JFrame;
4
5  public class TextFieldTest
6  {
7     public static void main( String[] args )
8     {
9        TextFieldFrame textFieldFrame = new TextFieldFrame();
10       textFieldFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       textFieldFrame.setSize( 350, 100 ); // set frame size
12       textFieldFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class TextFieldTest
```

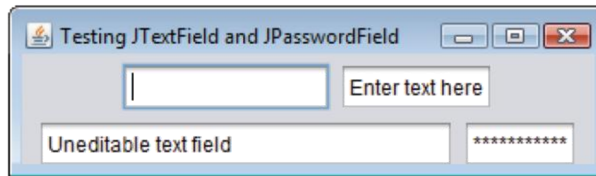
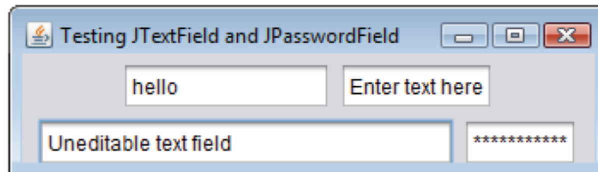
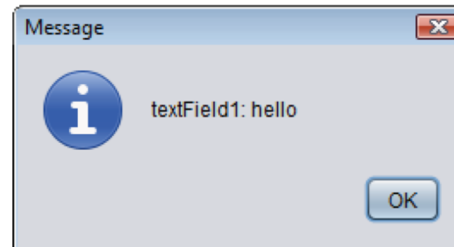
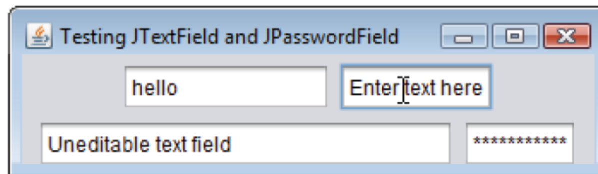


Fig. 14.10 | Test class for `TextFieldFrame`. (Part 1 of 3.)

(C) 2010 Pearson Education, Inc.
All rights reserved.







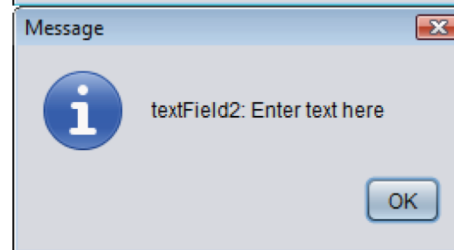
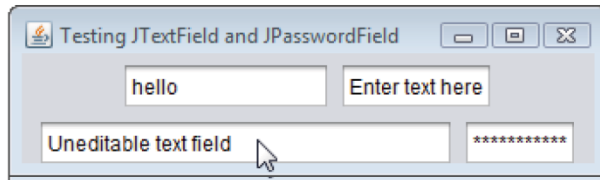
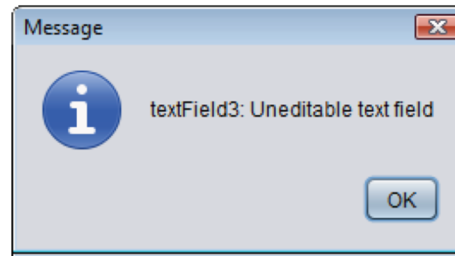
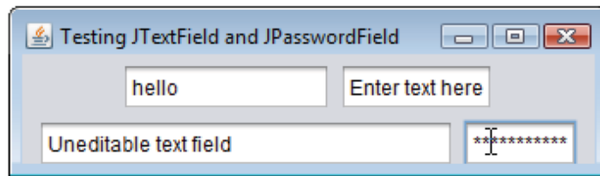


Fig. 14.10 | Test class for `TextFieldFrame`. (Part 2 of 3.)

(C) 2010 Pearson Education, Inc.
All rights reserved.







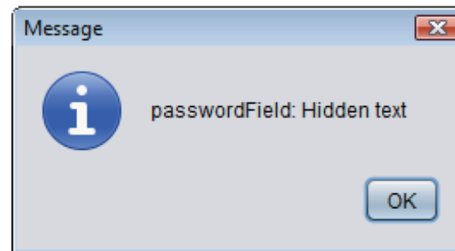


Fig. 14.10 | Test class for `TextFieldFrame`. (Part 3 of 3.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

Introduction to Event Handling (cont.)

- When the user types data into a `JTextField` or a `JPasswordField`, then presses *Enter*, an event occurs.
- You can type only in the text field that is "in focus."
- A component receives the focus when the user clicks the component.

(C) 2010 Pearson Education, Inc.
All rights reserved.

Introduction to Event Handling (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.

(C) 2010 Pearson Education, Inc.
All rights reserved.

Introduction to Event Handling (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.

(C) 2010 Pearson Education, Inc.
All rights reserved.

Software Engineering Observation 14.2

An inner class is allowed to directly access all of its top-level class's variables and methods.

(C) 2010 Pearson Education, Inc.
All rights reserved.

Introduction to Event Handling (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.
- A nested class that is `static` does not require an object of its top-level class and does not implicitly have a reference to an object of the top-level class.

(C) 2010 Pearson Education, Inc.
All rights reserved.

Introduction to Event Handling (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.

(C) 2010 Pearson Education, Inc.
All rights reserved.

Introduction to Event Handling (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.

(C) 2010 Pearson Education, Inc.
All rights reserved.

Introduction to Event Handling (cont.)

- When the user presses *Enter* in a `JTextField` or `JPasswordField`, an ActionEvent (package `java.awt.event`) occurs.
- Processed by an object that implements the interface ActionListener (package `java.awt.event`).
- To handle `ActionEvent`s, a class must implement interface `ActionListener` and declare method `actionPerformed`.
  - This method specifies the tasks to perform when an `ActionEvent` occurs.

(C) 2010 Pearson Education, Inc.
All rights reserved.

Software Engineering Observation 14.3

The event listener for an event must implement the appropriate event-listener interface.

(C) 2010 Pearson Education, Inc.
All rights reserved.

Common Programming Error 14.2

Forgetting to register an event-handler object for a particular GUI component's event type causes events of that type to be ignored.

(C) 2010 Pearson Education, Inc.
All rights reserved.

Introduction to Event Handling (cont.)

- Must register an object as the event handler for each text field.
- `addActionListener` registers an `ActionListener` object to handle `ActionEvent`s.
- After an event handler is registered the object listens for events.

(C) 2010 Pearson Education, Inc.
All rights reserved.

Introduction to Event Handling (cont.)

- The GUI component with which the user interacts is the event source.
- `ActionEvent` method `getSource` (inherited from class `EventObject`) returns a reference to the event source.
- `ActionEvent` method getActionCommand obtains the text the user typed in the text field that generated the event.
- `JPasswordField` method getPassword returns the password's characters as an array of type `char`.

(C) 2010 Pearson Education, Inc.
All rights reserved.

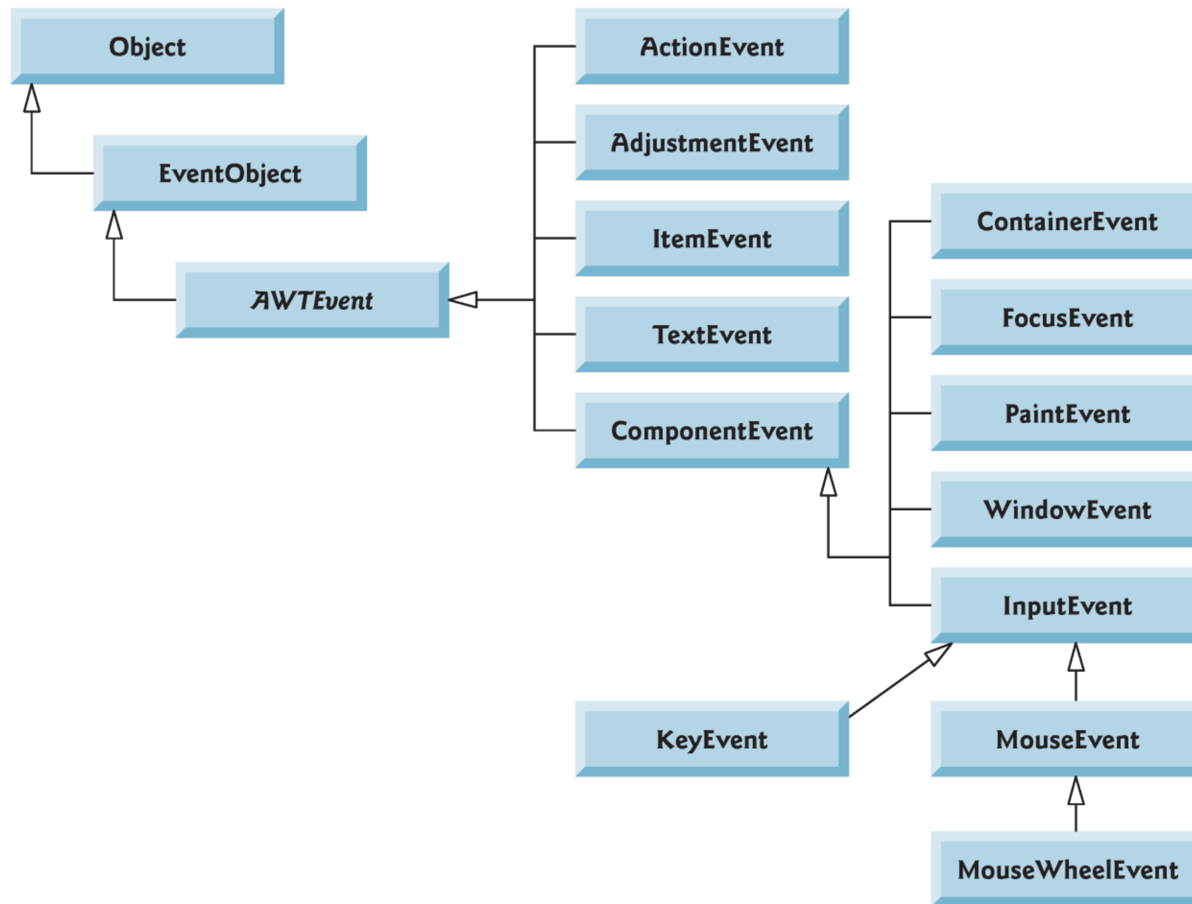


Fig. 14.11 | Some event classes of package `java.awt.event`.

(C) 2010 Pearson Education, Inc.
All rights reserved.

Common GUI Event Types and Listener Interfaces

- Figure 14.11 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`.

(C) 2010 Pearson Education, Inc.
All rights reserved.

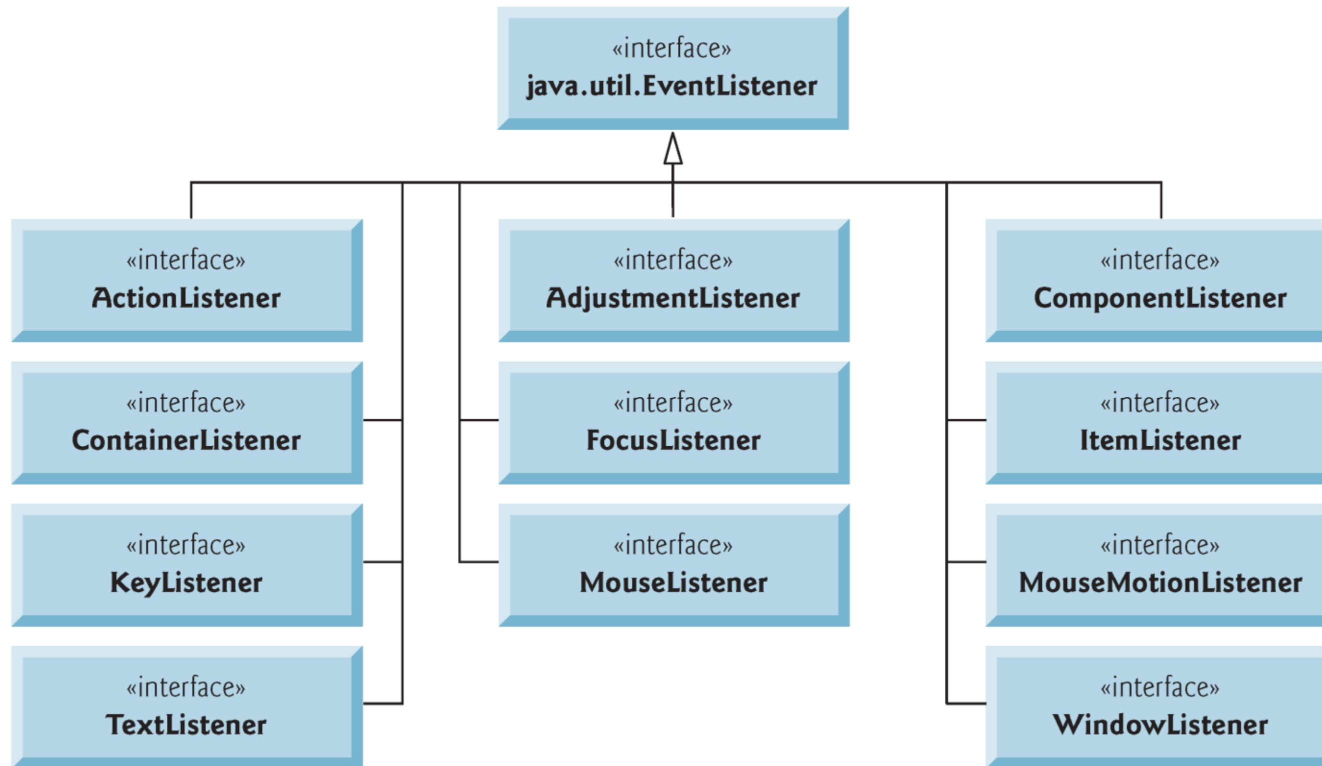


Fig. 14.12 | Some common event-listener interfaces of package `java.awt.event`.

(C) 2010 Pearson Education, Inc.
All rights reserved.

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. 14.12.
- Additional event-listener types that are specific to Swing components are declared in package `javax.swing.event`.

(C) 2010 Pearson Education, Inc.
All rights reserved.

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.

(C) 2010 Pearson Education, Inc.
All rights reserved.

How Event Handling Works

- How the event-handling mechanism works:
- Every `JComponent` has a variable `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.

(C) 2010 Pearson Education, Inc.
All rights reserved.

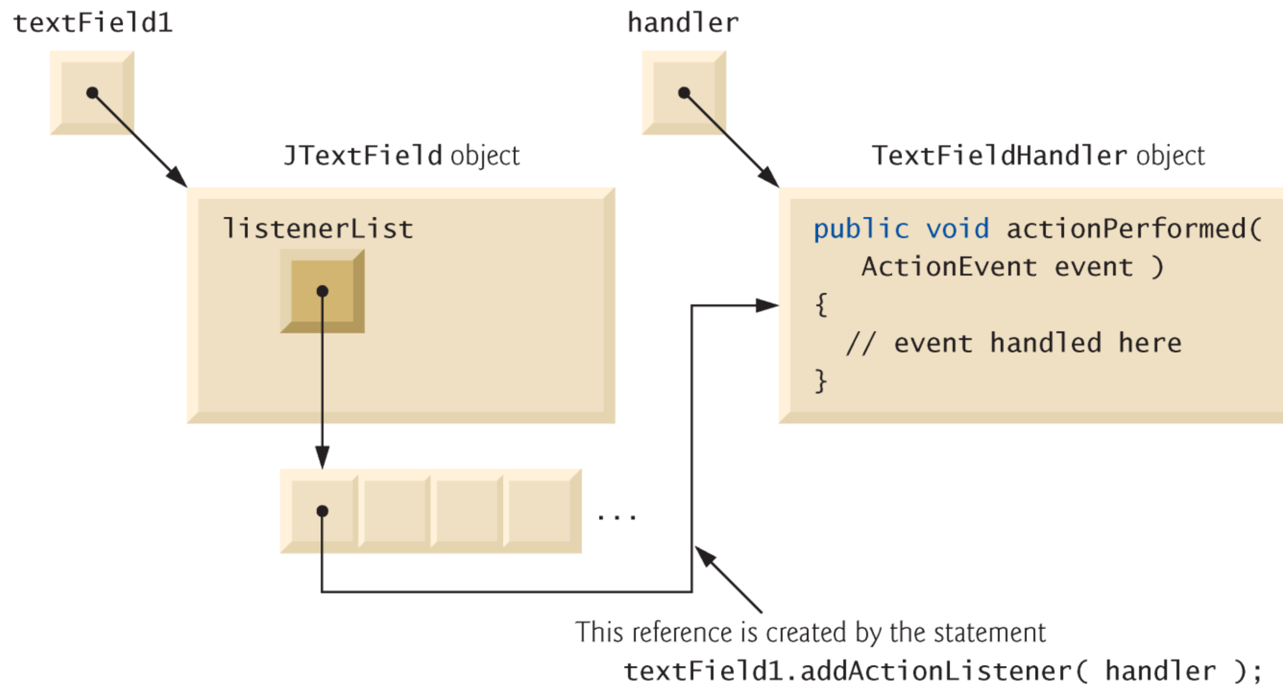


Fig. 14.13 | Event registration for `JTextField` `textField1`.

(C) 2010 Pearson Education, Inc.
All rights reserved.

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.

(C) 2010 Pearson Education, Inc.
All rights reserved.

How Event Handling Works (cont.)

- Each event type has one or more corresponding event-listener interfaces.
  - `ActionEvent`s are handled by `ActionListener`s
  - `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.

(C) 2010 Pearson Education, Inc.
All rights reserved.

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.

(C) 2010 Pearson Education, Inc.
All rights reserved.

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.

(C) 2010 Pearson Education, Inc.
All rights reserved.

`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. This event is always preceded by a call to `mousePressed`.
`public void mouseReleased( MouseEvent event )`
Called when a mouse button is released after being pressed. This event is 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.

Fig. 14.27 | `MouseListener` and `MouseMotionListener` interface methods. (Part 1 of 2.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

`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. This event is 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. 14.27 | `MouseListener` and `MouseMotionListener` interface methods. (Part 2 of 2.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

14.14 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` (`Mouse-Event`'s superclass) enable you to determine which mouse button the user clicked.

(C) 2010 Pearson Education, Inc.
All rights reserved.

Software Engineering Observation 14.7

Method calls to `mouseDragged` *are sent to the* `MouseMotionListener` *for the* `Component` *on which a mouse drag operation started. Similarly, the* `mouseReleased` *method call at the end of a drag operation is sent to the* `MouseListener` *for the* `Component` *on which the drag operation started.*

(C) 2010 Pearson Education, Inc.
All rights reserved.

14.14 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 `Mouse-Event`) contains methods that enable the event handler to obtain information about the amount of wheel rotation.

(C) 2010 Pearson Education, Inc.
All rights reserved.

```
1  // Fig. 14.28: MouseTrackerFrame.java
2  // Demonstrating mouse events.
3  import java.awt.Color;
4  import java.awt.BorderLayout;
5  import java.awt.event.MouseListener;
6  import java.awt.event.MouseMotionListener;
7  import java.awt.event.MouseEvent;
8  import javax.swing.JFrame;
9  import javax.swing.JLabel;
10 import javax.swing.JPanel;
11
12 public class MouseTrackerFrame extends JFrame
13 {
14    private JPanel mousePanel; // panel in which mouse events will occur
15    private JLabel statusBar; // label that displays event information
16
17    // MouseTrackerFrame constructor sets up GUI and
18    // registers mouse event handlers
19    public MouseTrackerFrame()
20    {
21       super( "Demonstrating Mouse Events" );
22
```

Fig. 14.28 | Mouse event handling. (Part 1 of 4.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

```
23        mousePanel = new JPanel(); // create panel
24        mousePanel.setBackground( Color.WHITE ); // set background color
25        add( mousePanel, BorderLayout.CENTER ); // add panel to JFrame
26
27        statusBar = new JLabel( "Mouse outside JPanel" );
28        add( statusBar, BorderLayout.SOUTH ); // add label to JFrame
29
30        // create and register listener for mouse and mouse motion events
31        MouseHandler handler = new MouseHandler();
32        mousePanel.addMouseListener( handler );
33        mousePanel.addMouseMotionListener( handler );
34     } // end MouseTrackerFrame constructor
35
36     private class MouseHandler implements MouseListener,
37        MouseMotionListener
38     {
39        // MouseListener event handlers
40        // handle event when mouse released immediately after press
41        public void mouseClicked( MouseEvent event )
42        {
43           statusBar.setText( String.format( "Clicked at [%d, %d]",
44              event.getX(), event.getY() ) );
45        } // end method mouseClicked
46
```

Object that handles both mouse events and mouse motion events.

An object of this class *is a* `MouseListener` and *is a* `MouseMotionListener`

Get the mouse coordinates at the time the click event occurred.

Fig. 14.28 | Mouse event handling. (Part 2 of 4.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

```
47         // handle event when mouse pressed
48         public void mousePressed( MouseEvent event )
49         {
50            statusBar.setText( String.format( "Pressed at [%d, %d]",
51               event.getX(), event.getY() ) );
52         } // end method mousePressed
53
54         // handle event when mouse released
55         public void mouseReleased( MouseEvent event )
56         {
57            statusBar.setText( String.format( "Released at [%d, %d]",
58               event.getX(), event.getY() ) );
59         } // end method mouseReleased
60
61         // handle event when mouse enters area
62         public void mouseEntered( MouseEvent event )
63         {
64            statusBar.setText( String.format( "Mouse entered at [%d, %d]",
65               event.getX(), event.getY() ) );
66            mousePanel.setBackground( Color.GREEN );
67         } // end method mouseEntered
68
```

Get the mouse coordinates at the time the pressed event occurred.

Get the mouse coordinates at the time the released event occurred.

Get the mouse coordinates at the time the entered event occurred then change the background to green.

Fig. 14.28 | Mouse event handling. (Part 3 of 4.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

```
69         // handle event when mouse exits area
70         public void mouseExited( MouseEvent event )
71         {
72            statusBar.setText( "Mouse outside JPanel" );
73            mousePanel.setBackground( Color.WHITE );
74         } // end method mouseExited
75
76         // MouseMotionListener event handlers
77         // handle event when user drags mouse with button pressed
78         public void mouseDragged( MouseEvent event )
79         {
80            statusBar.setText( String.format( "Dragged at [%d, %d]",
81               event.getX(), event.getY() ) );
82         } // end method mouseDragged
83
84         // handle event when user moves mouse
85         public void mouseMoved( MouseEvent event )
86         {
87            statusBar.setText( String.format( "Moved at [%d, %d]",
88               event.getX(), event.getY() ) );
89         } // end method mouseMoved
90      } // end inner class MouseHandler
91   } // end class MouseTrackerFrame
```

Change the background to white when the mouse exits the area.

Get the mouse coordinates at the time the dragged event occurred.

Get the mouse coordinates at the time the moved event occurred.

Fig. 14.28 | Mouse event handling. (Part 4 of 4.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

```
1  // Fig. 14.29: MouseTrackerFrame.java
2  // Testing MouseTrackerFrame.
3  import javax.swing.JFrame;
4
5  public class MouseTracker
6  {
7     public static void main( String[] args )
8     {
9        MouseTrackerFrame mouseTrackerFrame = new MouseTrackerFrame();
10       mouseTrackerFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       mouseTrackerFrame.setSize( 300, 100 ); // set frame size
12       mouseTrackerFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class MouseTracker
```

Fig. 14.29 | Test class for MouseTrackerFrame. (Part 1 of 2.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

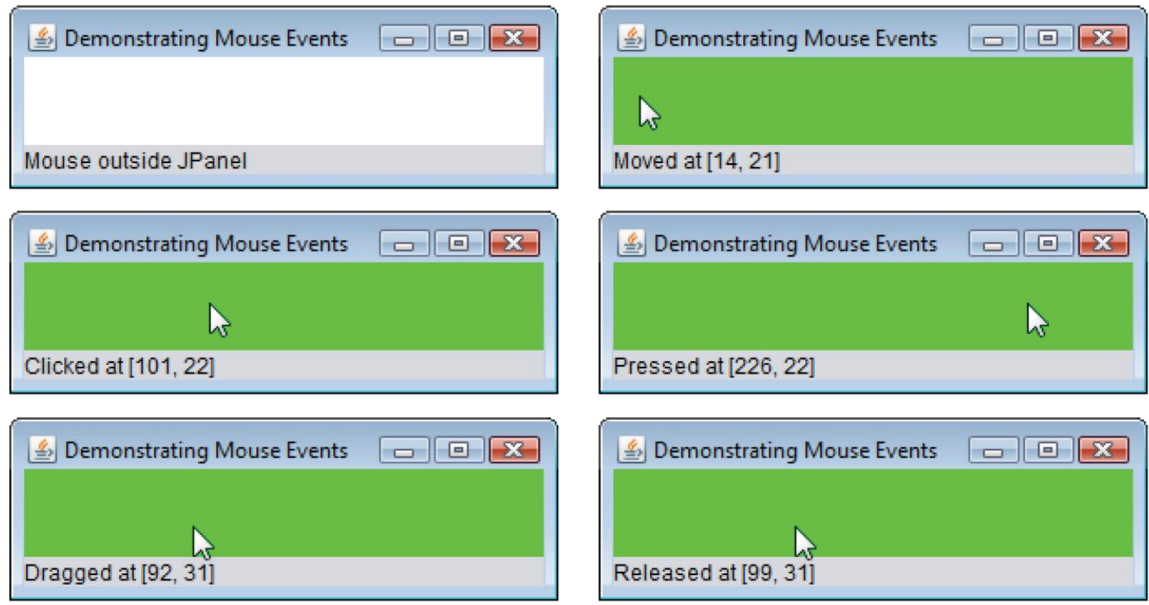


Fig. 14.29 | Test class for `MouseTrackerFrame`. (Part 2 of 2.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

14.14 Mouse Event Handling (cont.)

- BorderLayout arranges components into five regions: NORTH, SOUTH, EAST, WEST and CENTER.
- BorderLayout sizes the component in the CENTER to use all available space that is not occupied
- Methods addMouseListener and addMouseMotionListener register MouseListeners and MouseMotionListeners, respectively.
- MouseEvent methods getX and getY return the *x*- and *y*-coordinates of the mouse at the time the event occurred.

(C) 2010 Pearson Education, Inc.
All rights reserved.

14.15 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.

(C) 2010 Pearson Education, Inc.
All rights reserved.

Software Engineering Observation 14.8

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`*).*

(C) 2010 Pearson Education, Inc.
All rights reserved.

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

Fig. 14.30 | Event-adapter classes and the interfaces they implement in
package java.awt.event.

(C) 2010 Pearson Education, Inc.
All rights reserved.

```java
1  // Fig. 14.31: MouseDetailsFrame.java
2  // Demonstrating mouse clicks and distinguishing between mouse buttons.
3  import java.awt.BorderLayout;
4  import java.awt.event.MouseAdapter;
5  import java.awt.event.MouseEvent;
6  import javax.swing.JFrame;
7  import javax.swing.JLabel;
8
9  public class MouseDetailsFrame extends JFrame
10 {
11    private String details; // String that is displayed in the statusBar
12    private JLabel statusBar; // JLabel that appears at bottom of window
13
14    // constructor sets title bar String and register mouse listener
15    public MouseDetailsFrame()
16    {
17       super( "Mouse clicks and buttons" );
18
19       statusBar = new JLabel( "Click the mouse" );
20       add( statusBar, BorderLayout.SOUTH );
21       addMouseListener( new MouseClickHandler() ); // add handler
22    } // end MouseDetailsFrame constructor
23
```

Fig. 14.31 | Left, center and right mouse-button clicks. (Part 1 of 2.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

```
24      // inner class to handle mouse events
25      private class MouseClickHandler extends MouseAdapter
26      {
27         // handle mouse-click event and determine which button was pressed
28         public void mouseClicked( MouseEvent event )
29         {
30            int xPos = event.getX(); // get x-position of mouse
31            int yPos = event.getY(); // get y-position of mouse
32
33            details = String.format( "Clicked %d time(s)",
34               event.getClickCount() );
35
36            if ( event.isMetaDown() ) // right mouse button
37               details += " with right mouse button";
38            else if ( event.isAltDown() ) // middle mouse button
39               details += " with center mouse button";
40            else // left mouse button
41               details += " with left mouse button";
42
43            statusBar.setText( details ); // display message in statusBar
44         } // end method mouseClicked
45      } // end private inner class MouseClickHandler
46   } // end class MouseDetailsFrame
```

Adapter enables us to override the one method we use in this example.

Returns the number of mouse clicks. If you wait long enouogh between clicks, the count resets to 0.

Help determine which button the user pressed on the mouse.

Fig. 14.31 | Left, center and right mouse-button clicks. (Part 2 of 2.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

```java
1  // Fig. 14.32: MouseDetails.java
2  // Testing MouseDetailsFrame.
3  import javax.swing.JFrame;
4
5  public class MouseDetails
6  {
7     public static void main( String[] args )
8     {
9        MouseDetailsFrame mouseDetailsFrame = new MouseDetailsFrame();
10       mouseDetailsFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       mouseDetailsFrame.setSize( 400, 150 ); // set frame size
12       mouseDetailsFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class MouseDetails
```

Fig. 14.32 | Test class for MouseDetailsFrame. (Part 1 of 2.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

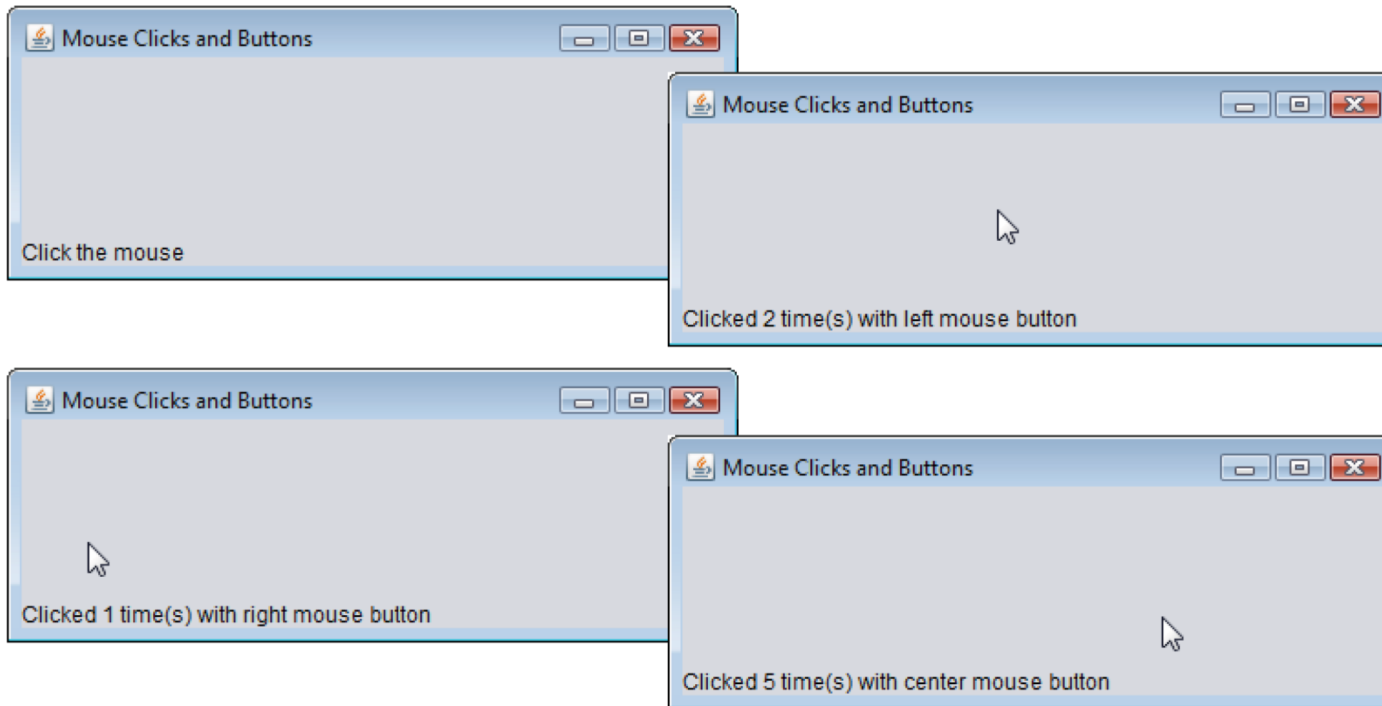


Fig. 14.32 | Test class for `MouseDetailsFrame`. (Part 2 of 2.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

Common Programming Error 14.4

If you extend an adapter class and misspell the name of the method you are overriding, your method simply becomes another method in the class. This is a logic error that is difficult to detect, since the program will call the empty version of the method inherited from the adapter class.

(C) 2010 Pearson Education, Inc.
All rights reserved.

14.15 Adapter Classes (cont.)

- A mouse can have one, two or three buttons.
- Class `MouseEvent` inherits several methods from `InputEvent` that can distinguish among mouse buttons or mimic a multibutton mouse with a combined keystroke and mouse-button click.
- Java assumes that every mouse contains a left mouse button.

(C) 2010 Pearson Education, Inc.
All rights reserved.

14.15 Adapter Classes (cont.)

- In the case of a one- or two-button mouse, a Java application assumes that the center mouse button is clicked if the user holds down the *Alt* key and clicks the left mouse button on a two-button mouse or the only mouse button on a one-button mouse.
- In the case of a one-button mouse, a Java application assumes that the right mouse button is clicked if the user holds down the *Meta* key (sometimes called the Command key or the "Apple" key on a Mac) and clicks the mouse button.

(C) 2010 Pearson Education, Inc.
All rights reserved.

InputEvent method	Description
isMetaDown()	Returns true when the user clicks the right mouse button on a mouse with two or three buttons. To simulate a right-mouse-button click on a one-button mouse, the user can hold down the *Meta* key on the keyboard and click the mouse button.
isAltDown()	Returns true when the user clicks the middle mouse button on a mouse with three buttons. To simulate a middle-mouse-button click on a one- or two-button mouse, the user can press the *Alt* key and click the only or left mouse button, respectively.

Fig. 14.33 | InputEvent methods that help distinguish among left-, center- and right-mouse-button clicks.

(C) 2010 Pearson Education, Inc.
All rights reserved.

14.15 Adapter Classes (cont.)

- The number of consecutive mouse clicks is returned by `MouseEvent` method `getClickCount`.
- Methods `isMetaDown` and `isAltDown` determine which mouse button the user clicked.

(C) 2010 Pearson Education, Inc.
All rights reserved.

14.16 `JPanel` Subclass for Drawing with the Mouse

- Use a `JPanel` as a dedicated drawing area in which the user can draw by dragging the mouse.
- Lightweight Swing components that extend class `JComponent` (such as `JPanel`) contain method `paintComponent`
  - called when a lightweight Swing component is displayed
- Override this method to specify how to draw.
  - Call the superclass version of `paintComponent` as the first statement in the body of the overridden method to ensure that the component displays correctly.

(C) 2010 Pearson Education, Inc.
All rights reserved.

14.16 JPanel Subclass for Drawing with the Mouse (cont.)

- JComponent support transparency.
  - To display a component correctly, the program must determine whether the component is transparent.
  - The code that determines this is in superclass JComponent's paintComponent implementation.
  - When a component is transparent, paintComponent will not clear its background
  - When a component is opaque, paintComponent clears the component's background
  - The transparency of a Swing lightweight component can be set with method setOpaque (a false argument indicates that the component is transparent).

(C) 2010 Pearson Education, Inc.
All rights reserved.

Look-and-Feel Observation 14.12

Most Swing GUI components can be transparent or opaque. If a Swing GUI component is opaque, its background will be cleared when its `paintComponent` *method is called. Only opaque components can display a customized background color.* `JPanel` *objects are opaque by default.*

(C) 2010 Pearson Education, Inc.
All rights reserved.

Error-Prevention Tip 14.1

In a `JComponent` *subclass's* `paintComponent` *method, the first statement should always call to the superclass's* `paintComponent` *method to ensure that an object of the subclass displays correctly.*

(C) 2010 Pearson Education, Inc.
All rights reserved.

Common Programming Error 14.5

If an overridden `paintComponent` *method does not call the superclass's version, the subclass component may not display properly. If an overridden* `paintComponent` *method calls the superclass's version after other drawing is performed, the drawing will be erased.*

(C) 2010 Pearson Education, Inc.
All rights reserved.

```
1  // Fig. 14.34: PaintPanel.java
2  // Using class MouseMotionAdapter.
3  import java.awt.Point;
4  import java.awt.Graphics;
5  import java.awt.event.MouseEvent;
6  import java.awt.event.MouseMotionAdapter;
7  import javax.swing.JPanel;
8
9  public class PaintPanel extends JPanel
10 {
11    private int pointCount = 0; // count number of points
12
13    // array of 10000 java.awt.Point references
14    private Point[] points = new Point[ 10000 ];
15
```

Fig. 14.34 | Adapter class used to implement event handlers. (Part 1 of 3.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

```
16      // set up GUI and register mouse event handler
17      public PaintPanel()
18      {
19         // handle frame mouse motion event
20         addMouseMotionListener(
21
22            new MouseMotionAdapter() // anonymous inner class
23            {
24               // store drag coordinates and repaint
25               public void mouseDragged( MouseEvent event )
26               {
27                  if ( pointCount < points.length )
28                  {
29                     points[ pointCount ] = event.getPoint(); // find point
30                     pointCount++; // increment number of points in array
31                     repaint(); // repaint JFrame
32                  } // end if
33               } // end method mouseDragged
34            } // end anonymous inner class
35         ); // end call to addMouseMotionListener
36      } // end PaintPanel constructor
37
```

Store points as user drags the mouse.

Request that this PaintPanel be repainted. Causes a call to paintComponent.

Fig. 14.34 | Adapter class used to implement event handlers. (Part 2 of 3.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

```
38     // draw ovals in a 4-by-4 bounding box at specified locations on window
39     public void paintComponent( Graphics g )
40     {
41        super.paintComponent( g ); // clears drawing area
42
43        // draw all points in array
44        for ( int i = 0; i < pointCount; i++ )
45           g.fillOval( points[ i ].x, points[ i ].y, 4, 4 );
46     } // end method paintComponent
47  } // end class PaintPanel
```

Draws a filled circle at the specified coordinates.

Fig. 14.34 | Adapter class used to implement event handlers. (Part 3 of 3.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

14.16 `JPanel` Subclass for Drawing with the Mouse (cont.)

- Class `Point` (package `java.awt`) represents an *x-y* coordinate.
  - We use objects of this class to store the coordinates of each mouse drag event.
- Class `Graphics` is used to draw.
- `MouseEvent` method `getPoint` obtains the `Point` where the event occurred.
- Method `repaint` (inherited from `Component`) indicates that a `Component` should be refreshed on the screen as soon as possible.

(C) 2010 Pearson Education, Inc.
All rights reserved.

Look-and-Feel Observation 14.13

Calling `repaint` *for a Swing GUI component indicates that the component should be refreshed on the screen as soon as possible. The background of the GUI component is cleared only if the component is opaque.* `JComponent` *method* `setOpaque` *can be passed a* `boolean` *argument indicating whether the component is opaque (*`true`*) or transparent (*`false`*).*

(C) 2010 Pearson Education, Inc.
All rights reserved.

14.16 `JPanel` Subclass for Drawing with the Mouse (cont.)

- `Graphics` method `fillOval` draws a solid oval.
  - Four parameters represent a rectangular area (called the bounding box) in which the oval is displayed.
  - The first two are the upper-left x-coordinate and the upper-left *y*-coordinate of the rectangular area.
  - The last two represent the rectangular area's width and height.
- Method `fillOval` draws the oval so it touches the middle of each side of the rectangular area.

(C) 2010 Pearson Education, Inc.
All rights reserved.

Look-and-Feel Observation 14.14

Drawing on any GUI component is performed with coordinates that are measured from the upper-left corner (0, 0) of that GUI component, not the upper-left corner of the screen.

(C) 2010 Pearson Education, Inc.
All rights reserved.

```
 1 // Fig. 14.35: Painter.java
 2 // Testing PaintPanel.
 3 import java.awt.BorderLayout;
 4 import javax.swing.JFrame;
 5 import javax.swing.JLabel;
 6
 7 public class Painter
 8 {
 9    public static void main( String[] args )
10    {
11       // create JFrame
12       JFrame application = new JFrame( "A simple paint program" );
13
14       PaintPanel paintPanel = new PaintPanel(); // create paint panel
15       application.add( paintPanel, BorderLayout.CENTER ); // in center
16
17       // create a label and place it in SOUTH of BorderLayout
18       application.add( new JLabel( "Drag the mouse to draw" ),
19          BorderLayout.SOUTH );
20
21       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
22       application.setSize( 400, 200 ); // set frame size
23       application.setVisible( true ); // display frame
24    } // end main
25 } // end class Painter
```

Creates the dedicated drawing area.

Attaches the dedicated drawing area to the center of the window.

Fig. 14.35 | Test class for PaintFrame. (Part 1 of 2.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

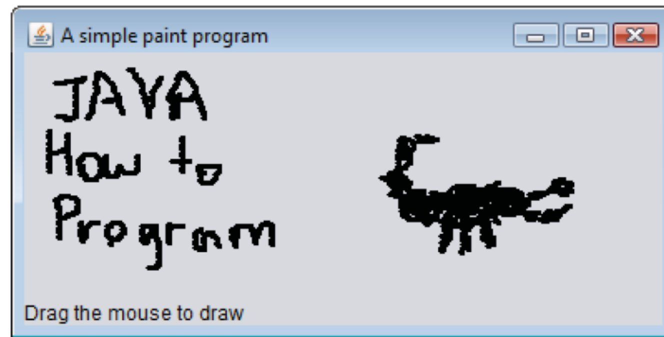


Fig. 14.35 | Test class for `PaintFrame`. (Part 2 of 2.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

14.17 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.

(C) 2010 Pearson Education, Inc.
All rights reserved.

```
1  // Fig. 14.36: KeyDemoFrame.java
2  // Demonstrating keystroke events.
3  import java.awt.Color;
4  import java.awt.event.KeyListener;
5  import java.awt.event.KeyEvent;
6  import javax.swing.JFrame;
7  import javax.swing.JTextArea;
8
9  public class KeyDemoFrame extends JFrame implements KeyListener
10 {
11    private String line1 = ""; // first line of textarea
12    private String line2 = ""; // second line of textarea
13    private String line3 = ""; // third line of textarea
14    private JTextArea textArea; // textarea to display output
15
16    // KeyDemoFrame constructor
17    public KeyDemoFrame()
18    {
19       super( "Demonstrating Keystroke Events" );
20
21       textArea = new JTextArea( 10, 15 ); // set up JTextArea
22       textArea.setText( "Press any key on the keyboard..." );
23       textArea.setEnabled( false ); // disable textarea
```

This class can handle its own KeyEvents.

Fig. 14.36 | Key event handling. (Part 1 of 3.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

```
24         textArea.setDisabledTextColor( Color.BLACK ); // set text color
25         add( textArea ); // add textarea to JFrame
26
27         addKeyListener( this ); // allow frame to process key events
28      } // end KeyDemoFrame constructor
29
30      // handle press of any key
31      public void keyPressed( KeyEvent event )
32      {
33         line1 = String.format( "Key pressed: %s",
34            KeyEvent.getKeyText( event.getKeyCode() ) ); // show pressed key
35         setLines2and3( event ); // set output lines two and three
36      } // end method keyPressed
37
38      // handle release of any key
39      public void keyReleased( KeyEvent event )
40      {
41         line1 = String.format( "Key released: %s",
42            KeyEvent.getKeyText( event.getKeyCode() ) ); // show released key
43         setLines2and3( event ); // set output lines two and three
44      } // end method keyReleased
45
```

Registers the object of this class as the event handler.

Gets text of pressed key.

Gets text of pressed key.

Fig. 14.36 | Key event handling. (Part 2 of 3.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

```
46      // handle press of an action key
47      public void keyTyped( KeyEvent event )
48      {
49         line1 = String.format( "Key typed: %s", event.getKeyChar() );
50         setLines2and3( event ); // set output lines two and three
51      } // end method keyTyped
52
53      // set second and third lines of output
54      private void setLines2and3( KeyEvent event )
55      {
56         line2 = String.format( "This key is %san action key",
57            ( event.isActionKey() ? "" : "not " ) );
58
59         String temp = KeyEvent.getKeyModifiersText( event.getModifiers() );
60
61         line3 = String.format( "Modifier keys pressed: %s",
62            ( temp.equals( "" ) ? "none" : temp ) ); // output modifiers
63
64         textArea.setText( String.format( "%s\n%s\n%s\n",
65            line1, line2, line3 ) ); // output three lines of text
66      } // end method setLines2and3
67   } // end class KeyDemoFrame
```

Gets text of pressed modifier keys.

Fig. 14.36 | Key event handling. (Part 3 of 3.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

```
1  // Fig. 14.37: KeyDemo.java
2  // Testing KeyDemoFrame.
3  import javax.swing.JFrame;
4
5  public class KeyDemo
6  {
7     public static void main( String[] args )
8     {
9        KeyDemoFrame keyDemoFrame = new KeyDemoFrame();
10       keyDemoFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       keyDemoFrame.setSize( 350, 100 ); // set frame size
12       keyDemoFrame.setVisible( true ); // display frame
13    } // end main
14 } // end class KeyDemo
```

Fig. 14.37 | Test class for KeyDemoFrame. (Part 1 of 2.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

Fig. 14.37 | Test class for `KeyDemoFrame`. (Part 2 of 2.)

(C) 2010 Pearson Education, Inc.
All rights reserved.

14.17 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.

(C) 2010 Pearson Education, Inc.
All rights reserved.

14.17 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.

(C) 2010 Pearson Education, Inc.
All rights reserved.