uCertify

Introduction to Java 11

uCertify

Lesson 12
Responding to User Input

Responding to User Input

- 12.1 Event Listeners
- 12.2 Working with Methods
- 12.3 Summary

Designing a Java program with a graphical user interface (GUI) isn't very useful if the user can't do anything to it. To make the program completely functional, you must make the interface receptive to user events.

12.1 Event Listeners

If a class wants to respond to a user event in Java, it must implement the interface that deals with that type of event. This interface is not the same thing as a GUI. It's an abstract type that defines a set of methods a class must implement.

Interfaces that handle user events are called *event listeners*.

Each listener handles a specific kind of event.

The `java.awt.event` package contains all the basic event listeners, as well as objects that represent specific events. These listener interfaces are some of the most useful:

- **ActionListener:** *Action events*, which are generated when a user performs an action on a component, such as clicking a button
- **AdjustmentListener:** *Adjustment events*, which are generated when a component is adjusted, such as when a scrollbar is moved
- **FocusListener:** *Keyboard focus events*, which are generated when a component such as a text field gains or loses the focus
- **ItemListener:** *Item events*, which are generated when an item such as a check box is changed
- **KeyListener:** *Keyboard events*, which occur when a user enters text by using the keyboard
- **MouseListener:** *Mouse events*, which are generated by mouse clicks, a mouse entering a component's area, and a mouse leaving a component's area
- **MouseMotionListener:** *Mouse movement events*, which track all movement by a mouse over a component
- **WindowListener:** *Window events*, which are generated when a window is maximized, minimized, moved, or closed

Listener Interfaces

Just as a Java class can implement multiple interfaces, a class that takes user input can implement as many listeners as needed. The `implements` keyword in the class declaration is followed by the name of the interface. If more than one interface has been implemented, their names are separated by commas.

The following class is declared to handle both action and text events:

```
public class Suspense extends JFrame implements ActionListener,
     TextListener {
    // body of class
}
```

To refer to these event listener interfaces in your programs, you can import them individually or use an `import` statement with a wildcard to make the entire package available:

```
import java.awt.event.*;
```

12.1 Event Listeners (Cont...)

uCertify

Setting Up Components

When you make a class an event listener, you have set up a specific type of event to be heard by that class. However, the event won't be heard unless you add a matching listener to the GUI component. That listener generates the events when the component is used.

After a component is created, you can call one (or more) of the following methods on the component to associate a listener with it:

- **addActionListener():** `JButton`, `JCheckBox`, `JComboBox`, `JTextField`, `JRadioButton`, and `JMenuItem` components
- **addFocusListener():** All Swing components
- **addItemListener():** `JButton`, `JCheckBox`, `JComboBox`, and `JRadioButton` components
- **addKeyListener():** All Swing components
- **addMouseListener():** All Swing components
- **addMouseMotionListener():** All Swing components
- **addTextListener():** `JTextField` and `JTextArea` components
- **addWindowListener():** `JWindow` and `JFrame` components

Methods associated to a Listener Interface

The following example creates a `JButton` object and associates an action event listener with it:

```
JButton zap = new JButton("Zap");
zap.addActionListener(this);
```

All the listener adding methods take one argument: the object that is listening for events of that kind. Using `this` indicates that the current class is the event listener. You could specify a different object, as long as its class implements the right listener interface.

12.1 Event Listeners (Cont...)

uCertify

Event-Handling Methods

When you associate an interface with a class, the class must contain methods that implement every method in the interface.

In the case of event listeners, the windowing system calls each method automatically when the corresponding user event takes place.

The **`ActionListener`** interface has only one method: **`actionPerformed()`**. All classes that implement **`ActionListener`** must have a method with the following structure:

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

If only one component in a program's GUI has a listener for action events, you know that this **`actionPerformed()`** method is called only in response to an event generated by that component. This makes it simpler to write the **`actionPerformed()`** method. All the method's code responds to that component's user event.

But when more than one component has an action event listener, you must use the method's **`ActionEvent`** argument to figure out which component was used and act accordingly in your program. You can use this object to discover details about the component that generated the event. **`ActionEvent`** and all other event objects are part of the **`java.awt.event`** package.

Every event-handling method is sent an event object of some kind. You can use the object's **`getSource()`** method to determine which component sent the event, as in the following example:

```
public void actionPerformed(ActionEvent event) {
    Object source = event.getSource();
}
```

12.1 Event Listeners (Cont…)

uCertify

Event-Handling Methods (Cont…)

The object returned by the **getSource()** method can be compared to components by using the equality operator (**==**). The following statements extend the preceding example to handle user clicks on buttons named **quitButton** and **sortRecords**:

```
if (source == quitButton) {
    quit();
}
if (source == sortRecords) {
    sort();
}
```

The **quit()** method is called if the **quitButton** object generated the event, and the **sort()** method is called if the **sortRecords** button generated the event.

Many event-handling methods call a different method for each kind of event or component. This makes an event-handling method easier to read. In addition, if a class has more than one event-handling method, each one can call the same methods to get work done.

Java's **instanceof** operator can be used in an event-handling method to determine the class of component that generated the event. The following example can be used in a program with one button and one text field, each of which generates an action event:

```
void actionPerformed(ActionEvent event) {
    Object source = event.getSource();
    if (source instanceof JTextField) {
        calculateScore();
    } else if (source instanceof JButton) {
        quit();
    }
}
```

If the event-generating component belongs to the **JTextField** class, the **calculateScore()** method is called. If the component belongs to **JButton**, the **quit()** method is called instead.

12.1 Event Listeners (Cont…)

Event-Handling Methods (Cont…)

Create a new Java File in NetBeans named **TitleBar.** Use the Listing 12.1

```
public class TitleBar extends JFrame implements ActionListener {
    JButton b1;
    JButton b2;

    public TitleBar() {
        super("Title Bar");
        setSize(330, 80);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        b1 = new JButton("Rosencrantz");
        b2 = new JButton("Guildenstern");
        b1.addActionListener(this);
        b2.addActionListener(this);
        FlowLayout flow = new FlowLayout();
        setLayout(flow);
        add(b1);
        add(b2);
        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent event) {
        Object source = event.getSource();
        if (source == b1) {
            setTitle("Rosencrantz");
        } else if (source == b2) {
            setTitle("Guildenstern");
        }
        repaint();
    }
```

Figure 12.1: The TitleBar application

uCertify

12.1 Event Listeners (Summary)

- When a user makes a class an event listener, a specific type of event to be heard by that class is to be specified.
- All the listener adding methods take only the object that is listening for events of that kind as an argument.
- The `ActionListener` interface has only the `actionPerformed()` method.
- The `ActionEvent` argument and all other event objects are part of the `java.awt.event` package.

12.2 Working with Methods

uCertify

Action Events

Action events occur when a user completes an action using components such as buttons, check boxes, menu items, text fields, and radio buttons.

A class must implement the **`ActionListener`** interface to handle these events. In addition, the **`addActionListener()`** method must be called on each component that should generate an action event—unless you want to ignore that component's action events.

The **`actionPerformed(ActionEvent)`** method is the only method of the **`ActionListener`** interface. It takes the following form:

```
public void actionPerformed(ActionEvent event) {
  // ...
}
```

In addition to the **`getSource()`** method, you can use the **`getActionCommand()`** method on the **`ActionEvent`** object to discover more information about the event's source.

By default, the action command is the text associated with the component, such as the label on a button. You also can set a different action command for a component by calling its **`setActionCommand(String)`** method. The *String* argument should be the action command's desired text.

The following statements create a button and a menu item and give both of them the action command **`"Sort Files"`**:

```
JButton sort = new JButton("Sort");
JMenuItem menuSort = new JMenuItem("Sort");
sort.setActionCommand("Sort Files");
menuSort.setActionCommand("Sort Files");
```

12.2 Working with Methods

uCertify

Focus Events

A focus event occurs when any component gains or loses input focus on a GUI. A component has the *focus* if it is the component that is active for keyboard input. If one of the fields has the focus (in a user interface with several editable text fields), the cursor blinks in the field. Any text entered goes into this component.

Focus applies to all components that can receive input. You can give a component the focus by calling its **`requestFocus()`** method with no arguments, as in this example:

```
JButton ok = new JButton("OK");
ok.requestFocus();
```

To handle a focus event, a class must implement the **`FocusListener`** interface, which has two methods: **`focusGained(FocusEvent)`** and **`focusLost(FocusEvent)`**. They take the following forms:

```
public void focusGained(FocusEvent event) {
  // ...
}
public void focusLost(FocusEvent event) {
  // ...
}
```

To determine which object gained or lost the focus, the **`getSource()`** method can be called on the **`FocusEvent`** object, sent as an argument to the two methods.

12.2 Working with Methods

Focus Events (Cont…)

Create a new Java File in NetBeans named **Calculator.** Use the Listing 12.2

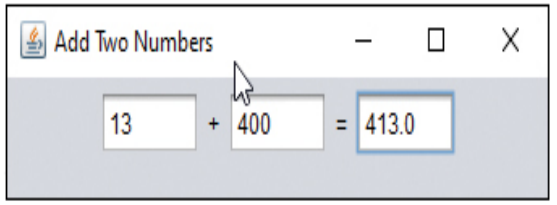


Figure 12.2: The Calculator application

In this application, the class implements the `FocusListener` interface so that focus listeners can be added to the first two text fields, `value1` and `value2`.

The `focusGained()` method is called whenever either of these fields gains the input focus (lines 38–48). In this method, the sum is calculated by adding the values in the other two fields. If either field contains an invalid value, such as a string, a `NumberFormatException` is thrown, and all three fields are reset to 0.

The `focusLost()` method accomplishes the same behavior by calling `focusGained()` with the focus event as an argument.

One thing to note about this application is that event-handling behavior is not required to collect numeric input in a text field. This is taken care of automatically by any component in which text input is received.

12.2 Working with Methods

Item Events

Item events occur when an item is selected or deselected on components such as buttons, check boxes, or radio buttons. A class must implement the `ItemListener` interface to handle these events.

The `itemStateChanged(ItemEvent)` method is the only method in the `ItemListener` interface. It takes the following form:

```
void itemStateChanged(ItemEvent event) {
  // ...
}
```

To determine in which item the event occurred, the `getItem()` method can be called on the *ItemEvent* object.

You also can see whether the item was selected or deselected by using the `getStateChange()` method. This method returns an integer that equals either the class variable `ItemEvent.DESELECTED` or `ItemEvent.SELECTED`.

Create a new Java File in NetBeans named **FormatChooser.** Use the Listing 12.3

This application extends the combo box example from Lesson 9, "Creating a Graphical User Interface." Figure 12.3 shows how it looks after a choice has been made.

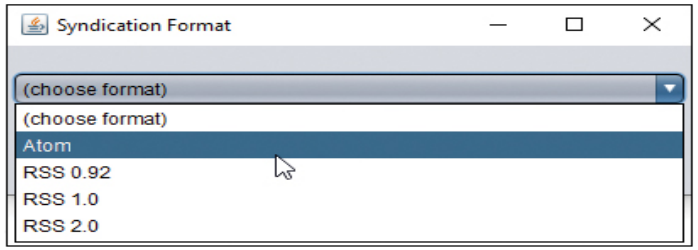


Figure 12.3: The output of the FormatChooser application

The application creates a combo box from an array of strings and adds an item listener to the component (lines 24–27). Item events are received by the `itemStateChanged(ItemEvent)` method (lines 34–39), which changes a label's text based on the index number of the selected item. Index 1 corresponds with Atom, 2 with RSS 0.92, 3 with RSS 1.0, and 4 with RSS 2.0.

12.2 Working with Methods

Key Events

A key event occurs when a keyboard key is pressed. Any component can generate a key event, and a class must implement the **KeyListener** interface to support these events.

The **KeyListener** interface has three methods: **keyPressed(*KeyEvent*)**, **keyReleased(*KeyEvent*)**, and **keyTyped(*KeyEvent*)**. They take the following forms:

```
public void keyPressed(KeyEvent event) {
    // ...
}
public void keyReleased(KeyEvent event) {
    // ...
}
public void keyTyped(KeyEvent event) {
    // ...
}
```

KeyEvent's **getKeyChar()** method returns the character of the key associated with the event. If no Unicode character can be represented by the key, **getKeyChar()** returns a character value equal to the class variable **KeyEvent.CHAR_UNDEFINED**.

For a component to generate key events, it must be able to receive the input focus. Text fields, text areas, and other components that accept keyboard input support this capability automatically. For other components, such as labels and panels, the **setFocusable(*boolean*)** method should be called with the argument **true**, as in the following code:

```
JPanel pane = new JPanel();
pane.setFocusable(true);
```

12.2 Working with Methods

Mouse Events

A mouse event is generated by a mouse click, a mouse entering a component's area, or a mouse leaving the area. Any component can generate these events, which are implemented by a class through the `MouseListener` interface, which has five methods:

- `mouseClicked(MouseEvent)`
- `mouseEntered(MouseEvent)`
- `mouseExited(MouseEvent)`
- `mousePressed(MouseEvent)`
- `mouseReleased(MouseEvent)`

Each method has the same basic form as `mouseReleased(MouseEvent)`:

```
public void mouseReleased(MouseEvent event) {
    // ...
}
```

The following methods can be used on `MouseEvent` objects:

- **getClickCount()**: Returns, as an integer, the number of times the mouse was clicked
- **getPoint()**: Returns, as a `Point` object, the (x, y) coordinates within the component where the mouse was clicked
- **getX()**: Returns the x position
- **getY()**: Returns the y position

12.2 Working with Methods

Mouse Motion Events

A mouse motion event occurs when the mouse is moved over a component. As with other mouse events, any component can generate mouse motion events. A class must implement the `MouseMotionListener` interface to support them.

The `MouseMotionListener` interface has two methods: `mouseDragged(MouseEvent)` and `mouseMoved(MouseEvent)`. They take the following forms:

```
public void mouseDragged(MouseEvent event) {
    // ...
}
public void mouseMoved(MouseEvent event) {
    // ...
}
```

Unlike the other event listener interfaces you have dealt with up to this point, `MouseMotionListener` does not have its own event type. Instead, `MouseEvent` objects are used. Because of this, you can call the same methods you would call for mouse events: `getClick()`, `getPoint()`, `getX()`, and `getY()`.

The next project you will undertake demonstrates how to detect and respond to mouse events. The `MousePrank` application, shown in Listing 12.4, consists of two classes, `MousePrank` and `PrankPanel`, that implement a user interface button that tries to avoid being clicked.

Create a new Java File in NetBeans named **MousePrank.** Use the Listing 12.4

12.2 Working with Methods

Windows Events

A window event occurs when a user opens or closes a window object, such as a `JFrame` or `JWindow` object. Any component can generate these events, and a class must implement the `WindowListener` interface to support them.

The `WindowListener` interface has seven methods:

- `windowActivated(WindowEvent)`
- `windowClosed(WindowEvent)`
- `windowClosing(WindowEvent)`
- `windowDeactivated(WindowEvent)`
- `windowDeiconified(WindowEvent)`
- `windowIconified(WindowEvent)`
- `windowOpened(WindowEvent)`

They all have the same form as the `windowOpened()` method:

```
public void windowOpened(WindowEvent event) {
    // body of method
}
```

The `windowClosing()` and `windowClosed()` methods are similar, but one is called as the window is closing, and the other is called after it is closed. In fact, you can take action in a `windowClosing()` method to stop the window from being closed.

12.2 Working with Methods

Using Adapter Classes

A Java class that implements an interface must include all its methods, even if it doesn't plan to do anything in response to some of them.

This requirement can make it necessary to add a lot of empty methods when you're working with an event-handling interface such as `WindowListener`, which has seven methods.

As a convenience, Java offers *adapters*, which are Java classes that contain empty do-nothing implementations of specific interfaces. By subclassing an adapter class, you can implement only the event-handling methods you need by overriding those methods. The rest inherit those do-nothing methods.

The `java.awt.event` package includes `FocusAdapter`, `KeyAdapter`, `MouseAdapter`, `MouseMotionAdapter`, and `WindowAdapter`. They correspond to the expected listeners for focus, keyboard, mouse, mouse motion, and window events.

Create a new Java File in NetBeans named **KeyChecker.** Use the Listing 12.5

The `KeyChecker` application is implemented as a main class of the same name and a `KeyMonitor` helper class.

`KeyMonitor` is a subclass of `KeyAdapter`, an adapter class for keyboard events that implements the `KeyListener` interface. In lines 47–50, the `keyTyped` method overrides the same method in `KeyAdapter`, which does nothing.

When a key is pressed, the key is discovered by calling the `getKeyChar()` method of the user event object. This key becomes the text of the `keyLabel` label in the `KeyChecker` class. This application is shown in Figure 12.5.

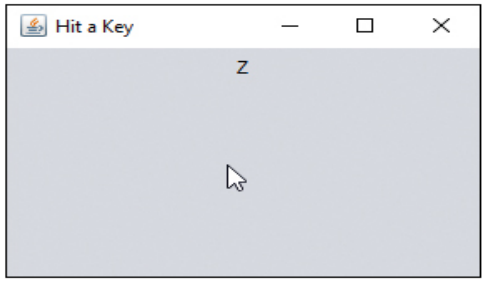


Figure 12.5: The running KeyChecker application

12.2 Working with Methods - Summary

- Action events occur when a user completes an action using components, such as buttons, check boxes, menu items, text fields, and radio buttons.
- A class must implement the `ActionListener` interface to handle action events.
- A focus event occurs when any component gains or loses input focus on a graphical user interface (GUI).
- A user can give a component a focus by calling its `requestFocus()` method with no arguments.
- Item events occur when an item is selected or deselected on components, such as buttons, check boxes, or radio buttons.

12.2 Working with Methods - Summary (continued)

- A class must implement the `ItemListener` interface to handle item events.
- A key event occurs when a keyboard key is pressed.
- A class must implement the `KeyListener` interface to support key events.
- A mouse event is generated by a mouse click, a mouse entering a component's area, or a mouse leaving the area.
- A class must implement the `MouseListener` interface to handle mouse events.
- A mouse motion event occurs when a mouse is moved over a component.

uCertify

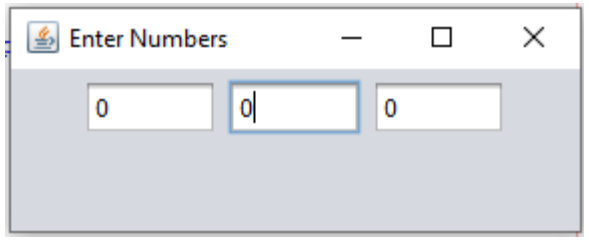
12.2 Working with Methods - Summary (continued)

- A class must implement the `MouseMotionListener` interface to support mouse motion events.
- A window event occurs when a user opens or closes a window object, such as a `JFrame` or `JWindow` object.
- A class must implement the `WindowListener` interface to handle window events.
- A Java class that implements an interface must include all its methods even if it doesn't plan to do anything in response to some of them.
- Java offers adapters, which are Java classes that contain empty do-nothing implementations of specific interfaces.

12.3 Summary

In this presentation, we have learned about:

- Adding Event handling to a GUI in Swing
- Learning which component generated an event
- Working with listener interfaces and event classes
- Check the Exercises in the Summary
  - Exercise 1 - **PossitiveFrame** create an application that uses **FocusListener** to ensure that the text field's value is multiplied by -1 and is redisplayed whenever a user changes it to a negative value



  - Exercise 2 – **Calculator2** create a calculator that adds or subtracts the contents of two text fields whenever the appropriate button is clicked and displays the result as a label