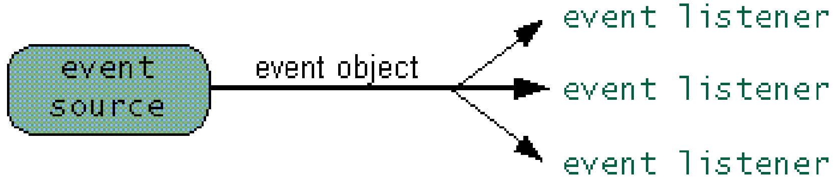
Event Handling

Ji Zhenyan & Åke Malmberg

Event Handling

- Every time the user types a character or pushes a mouse button, an event occurs.
- Each event is represented by an object that gives information about the event and identifies the event source.
- each event source can have multiple listeners registered on it.
- a single listener can register with multiple event sources.
-



Implement an Event Handler

- declare the event handler class, specify that the class either implements a listener interface or extends a class that implements a listener interface.

public class MyClass implements ActionListener {

- registers an instance of the event handler class

as a listener upon one or more components.

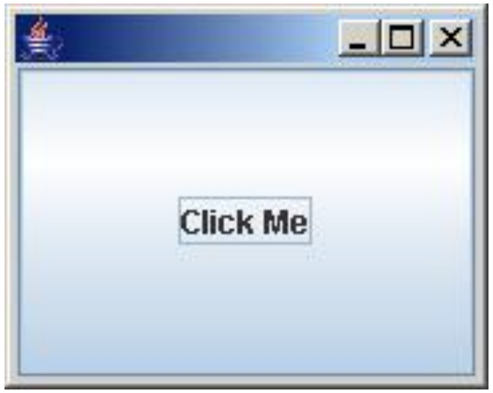
someComponent.addActionListener(instanceOfMyClass);

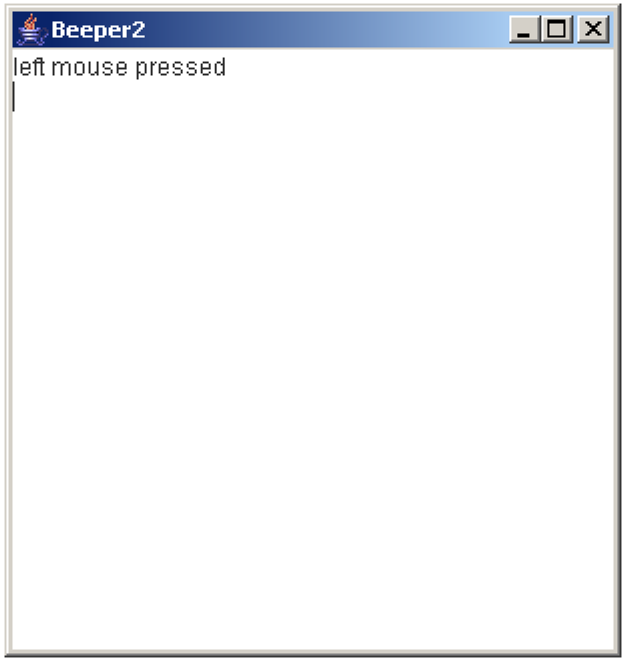
- implements the methods in the listener interface.

public void actionPerformed(ActionEvent e) { *...//code that reacts to the action...* }

Implement an Event Handler

- Beeper.java
- Beeper2.java





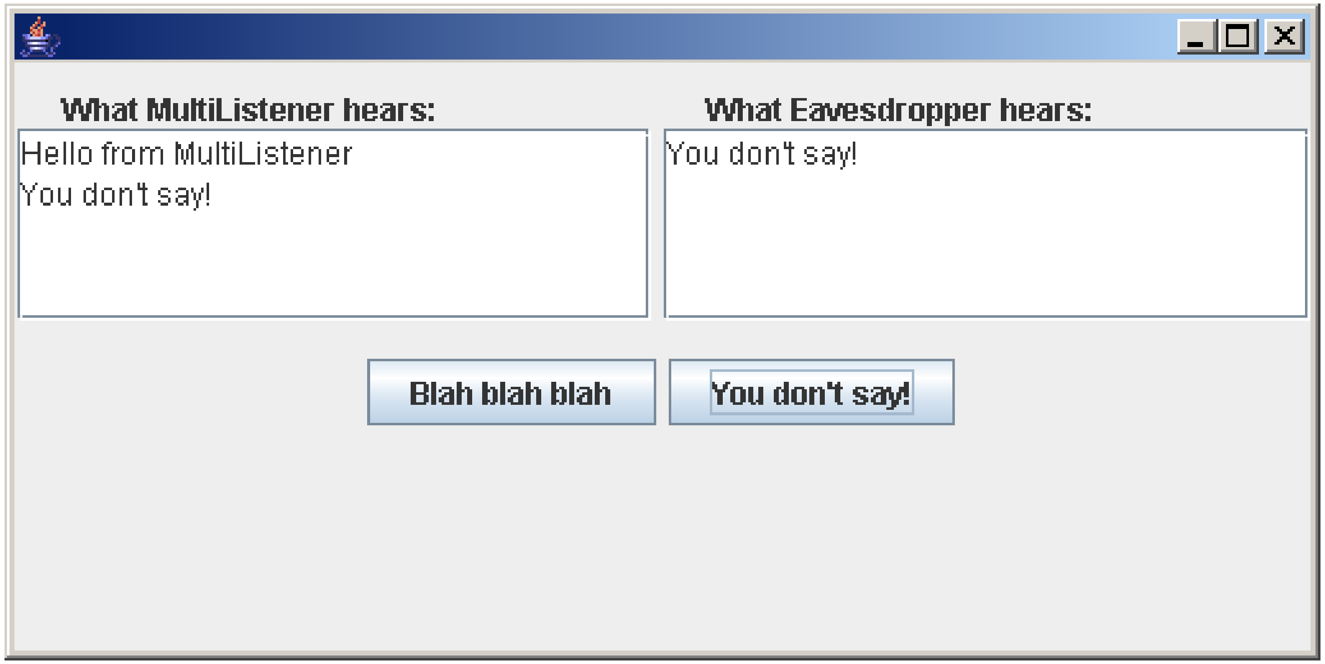
```
left mouse pressed
```

Implement an Event Handler

- a program might have a single listener for all events from all sources.
- A program can have more than one listener for a single kind of event from a single event source.

Implement an Event Handler

MultiListener.java



```
Hello from MultiListener
You don't say!
```

```
You don't say!
```

Using Adapters and Inner Classes to Handle Events

- Most listener interfaces contain more than one method.
- the MouseListener interface contains five methods: mousePressed, mouseReleased, mouseEntered, mouseExited, and mouseClicked. Even if you care only about mouse clicks, if your class directly implements MouseListener, then you must implement all five MouseListener methods.
- the resulting collection of empty method bodies can make code harder to read and maintain.
- the API generally includes an *adapter* class for each listener interface with more than one method.
- To use an adapter, you create a subclass of it.

Worse!!

- public class MyClass implements MouseListener {

```
        ...
        someObject.addMouseListener(this);
        ...
        /* Empty method definition. */
        public void mousePressed(MouseEvent e) { }
        public void mouseReleased(MouseEvent e) { }
        public void mouseEntered(MouseEvent e) { }
        public void mouseExited(MouseEvent e) { }
        public void mouseClicked(MouseEvent e) {
                ...//Event handler implementation goes
here...
        }
}
```

Better!!

- public class MyClass extends MouseAdapter {

```
    ...
    someObject.addMouseListener(this);
    ...
    public void mouseClicked(MouseEvent e) {
        ...
        //Event handler implementation goes here
        ...
    }
}
```

Anonymous class

- An example of using an anonymous inner class.

```
public class MyClass extends Applet {
                    ...
                    someObject.addMouseListener(new
   MouseAdapter() {
                     public void mouseClicked(MouseEvent e)
   { ...//Event handler implementation goes here... } }); ...
   } }
```

- You can tell what kinds of events a component can fire by looking at the kinds of event listeners you can register on it. (JButton, JTree)

Listeners that All Swing Components Support

Because all Swing components descend from the AWT Component class

- **component listener**
  - Listens for changes in the component's size, position, or visibility.
- **focus listener**
  - Listens for whether the component gained or lost the ability to receive keyboard input.
- **key listener**
  - Listens for key presses; key events are fired only by the component that has the current keyboard focus.
- **mouse events**
  - Listens for mouse clicks and mouse movement into or out of the component's drawing area.
- **mouse-motion events**
  - Listens for changes in the cursor's position over the component.

- a component fires only those events for which listeners have registered on it.
- For example, if a mouse listener is registered on a particular component, but the component has no other listeners, then the component will fire only mouse events--no component, focus, key, or mouse-motion events.

Action Listener

- When the user clicks a button , chooses a menu item or presses Return in a text field , an action event occurs.
- Action listeners are probably the easiest -- and most common -- event handlers to implement.
- The ActionListener interface contains a method:
- **void actionPerformed(ActionEvent)**
  - The ActionEvent class defines two useful methods:
- **String getActionCommand()**
  - Returns the string associated with this action. Most objects that can fire action events support a method called setActionCommand that lets you set this string. If you don't set the action command explicitly, then it's generally the text displayed in the component. For objects with multiple items, and thus multiple possible actions, the action command is generally the name of the selected item.

- **int getModifiers()**
  - Returns an integer representing the modifier keys the user was pressing when the action event occurred. You can use the ActionEvent-defined constants SHIFT_MASK, CTRL_MASK, META_MASK, and ALT_MASK to determine which keys were pressed. For example, if the user Shift-selects a menu item, then the following expression is nonzero: actionEvent.getModifiers() & ActionEvent.SHIFT_MASK
- getSource() method, which ActionEvent inherits from EventObject .

Caret Listener

- Caret events occur when the caret in a text component moves or when the selection in a text component changes. You can attach a caret listener to an instance of any JTextComponent subclass with the addCaretListener method.
- The CaretListener interface has just one method:
- **void caretUpdate(CaretEvent)**
  - Called when the caret in the listened-to component moves or when the selection in the listened-to component changes.
- To get the text component that fired the event, use the getSource method which CaretEvent inherits from EventObject .
- The CaretEvent class defines two useful methods:
- **int getDot()**
  - Returns the current location of the caret. If text is selected, the caret marks one end of the selection.
- **int getMark()**
  - Returns the other end of the selection. If nothing is selected, the value returned by this method is equal to the value returned by getDot. Note that the dot is not guaranteed to be less than the mark.

Change Listener

- Change events occur whenever a component changes state. For example, a button fires a change event every time the button is pressed. Although nothing's stopping you from registering for change events on a button, most programs don't need to do so.
- Two Swing components rely on change events for basic functionality -- sliders and color choosers. To learn when the value in a slider changes, you need to register a change listener. Similarly, you need to register a change listener on a color chooser to be informed when the user chooses a new color.
- The ChangeListener interface has just one method, so it has no corresponding adapter class.
- **void stateChanged(ChangeEvent)**

ComponentListener

- One or more component events are fired by a Component object just after the component is hidden, made visible, moved, or resized. ComponentListener interface, adapter class, ComponentAdapter , contain four methods:
- **void componentHidden(ComponentEvent)**

  Called after the listened-to component is hidden as the result of the setVisible method being called.
- **void componentMoved(ComponentEvent)**

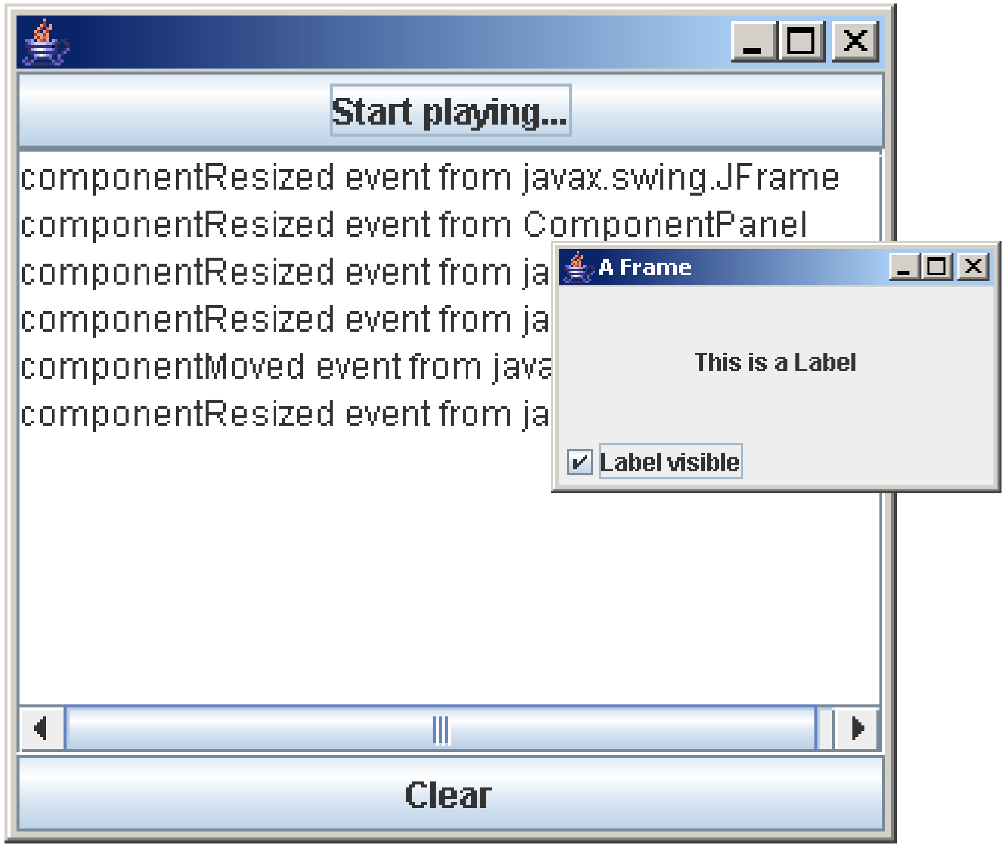
  Called after the listened-to component moves, relative to its container. For example, if a window is moved, the window fires a component-moved event, but the components it contains do not.
- **void componentResized(ComponentEvent)**

  Called after the listened-to component's size (rectangular bounds) changes.
- **void componentShown(ComponentEvent)**

  Called after the listened-to component becomes visible as the result of the setVisible method being called.

- The ComponentEvent class defines the following useful method:
- **Component getComponent()**
  - Returns the component that fired the event. You can use this instead of the getSource method.

ComponentEventDemo.java

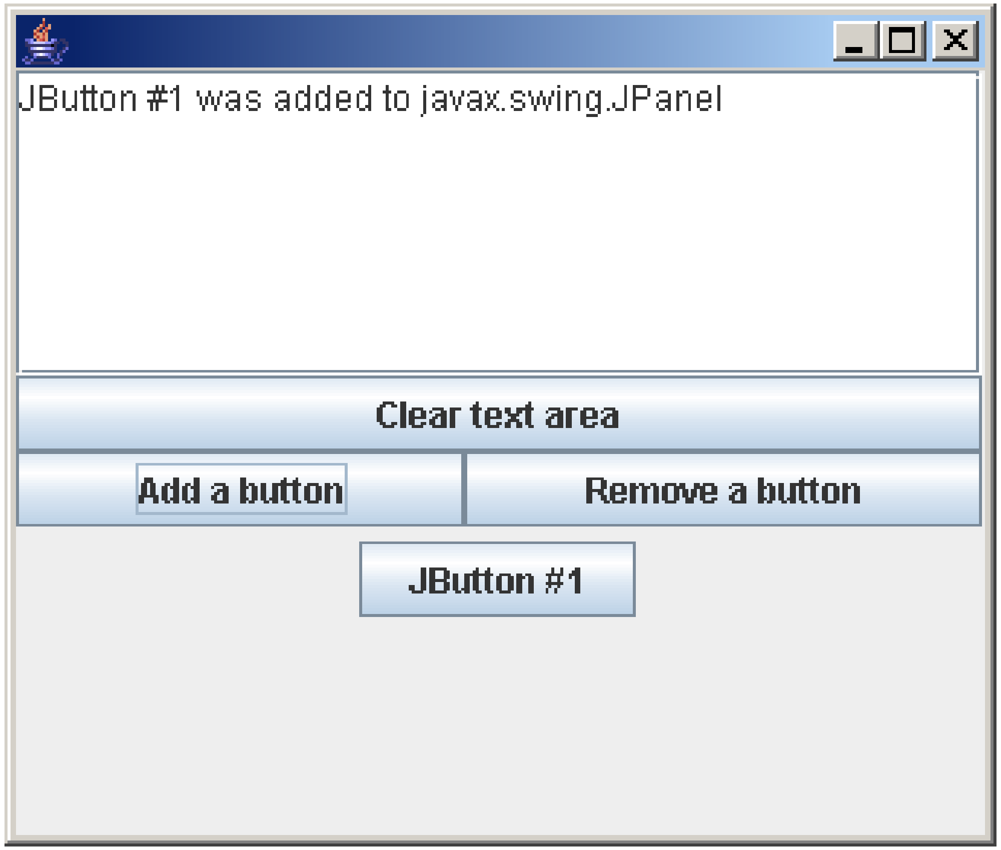


```
componentResized event from javax.swing.JFrame
componentResized event from ComponentPanel
componentResized event from ja
componentResized event from ja
componentMoved event from java
componentResized event from ja
```

Container Listener

- Container events are fired by a Container just after a component is added to or removed from the container.
- The ContainerListener interface and adapter class, ContainerAdapter, contain two methods:
- **void componentAdded(ContainerEvent)**
  - Called just after a component is added to the listened-to container.
- **void componentRemoved(ContainerEvent)**
  - Called just after a component is removed from the listened-to container.
- The ContainerEvent class defines two useful methods:
- **Component getChild()**
  - Returns the component whose addition or removal triggered this event.
- **Container getContainer()**
  - Returns the container that fired this event. You can use this instead of the getSource method.

ContainerEventDemo.java

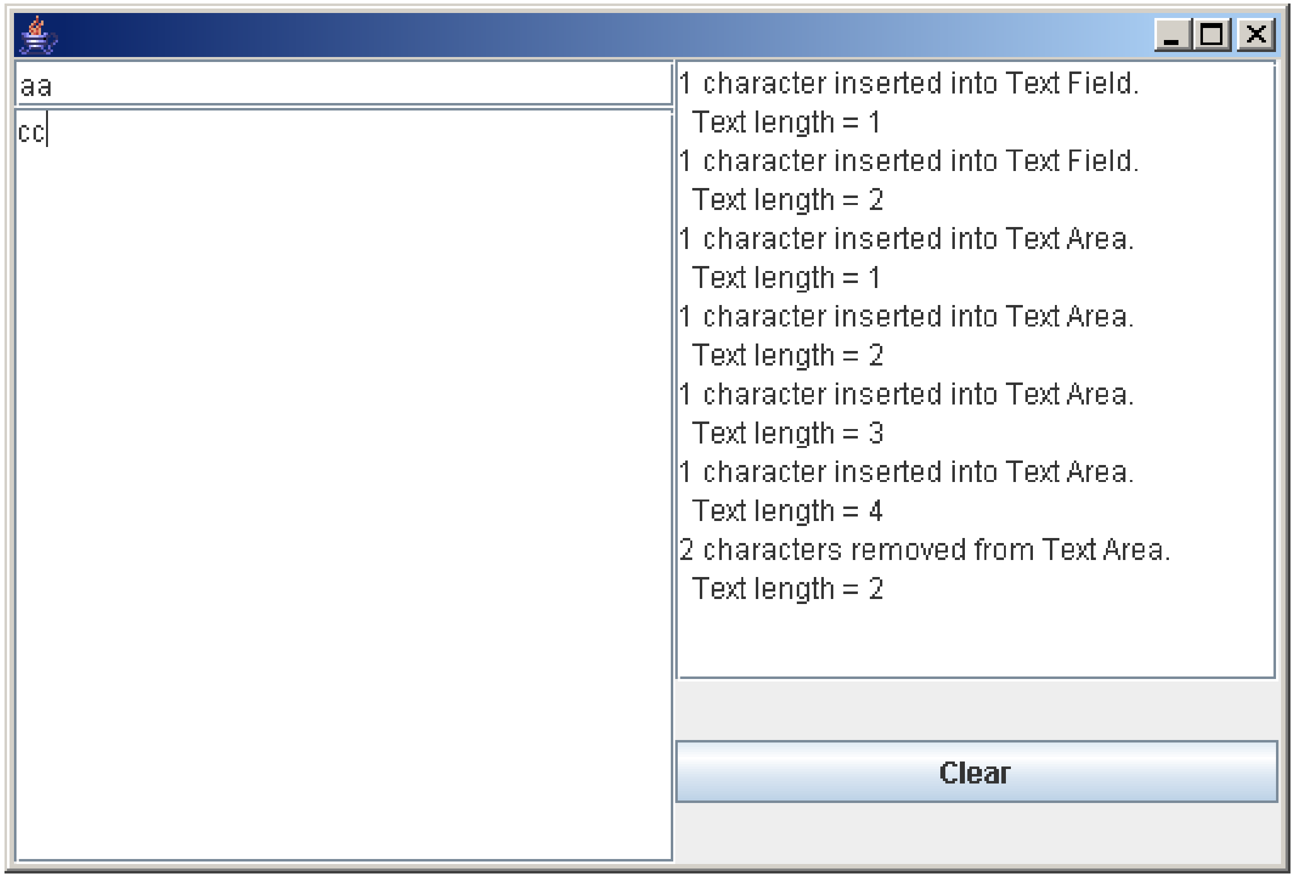

JButton #1 was added to javax.swing.JPanel

Document Listener

- A Swing text component uses a Document to hold and edit its text. Document events occur when the content of a document changes in any way. You attach a document listener to a text component's document, rather than to the text component itself.
- The DocumentListener interface contains three methods:
- **void changedUpdate(DocumentEvent)**
  - Called when the style of some of the text in the listened-to document changes. This sort of event is fired only from a StyledDocument -- a PlainDocument does not fire these events.
- **void insertUpdate(DocumentEvent)**
  - Called when text is inserted into the listened-to document.
- **void removeUpdate(DocumentEvent)**
  - Called when text is removed from the listened-to document.
- Each document event method has a single parameter: an instance of a class that implements the DocumentEvent interface. Typically, the object passed into this method will be an instance of DefaultDocumentEvent which is defined in AbstractDocument.

- To get the document that fired the event, you can use DocumentEvent's getDocument method. Note that as an interface, DocumentEvent does not inherit from EventObject. Thus, it does not inherit the getSource method.
- In addition to getDocument, the DocumentEvent interface requires these methods:
- int getLength()
  - Returns the length of the change.
- int getOffset()
  - Returns the location within the document of the first character changed.
- ElementChange getChange(Element)
  - Returns details about what elements in the document have changed and how. ElementChange is an interface defined within the DocumentEvent interface.
- EventType getType()
  - Returns the type of change that occurred. EventType is a class defined within the DocumentEvent interface that enumerates the possible changes that can occur on a document: insert text, remove text, and change text style.

DocumentEventDemo.java

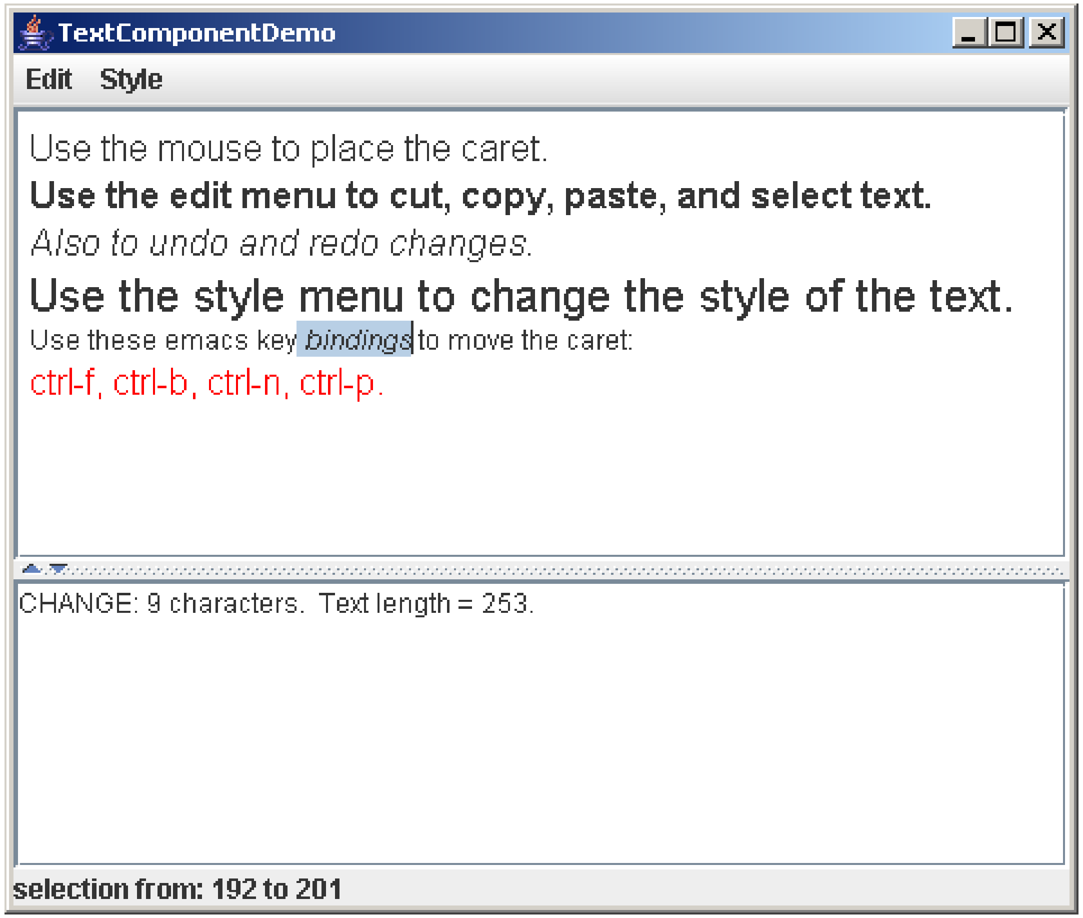


```
aa
```

```
cc
```

```
1 character inserted into Text Field.
  Text length = 1
1 character inserted into Text Field.
  Text length = 2
1 character inserted into Text Area.
  Text length = 1
1 character inserted into Text Area.
  Text length = 2
1 character inserted into Text Area.
  Text length = 3
1 character inserted into Text Area.
  Text length = 4
2 characters removed from Text Area.
  Text length = 2
```

TextComponentDemo.java



Use the mouse to place the caret.
Use the edit menu to cut, copy, paste, and select text.
Also to undo and redo changes.
Use the style menu to change the style of the text.
Use these emacs key *bindings* to move the caret:
ctrl-f, ctrl-b, ctrl-n, ctrl-p.

```
CHANGE: 9 characters.  Text length = 253.
```

selection from: 192 to 201

Mouse Listener

- Mouse events occur when the cursor enters or exits a component's on-screen area and when the user presses or releases the mouse button.
- If your program needs to detect both mouse events and mouse-motion events, you can use Swing's convenient MouseInputAdapter class, which implements both MouseListener and MouseMotionListener.
- The MouseListener interface and adapter class, MouseAdapter, contain methods:
- **void mouseClicked(MouseEvent)**
  - Called just after the user clicks the listened-to component.
- **void mouseEntered(MouseEvent)**
  - Called just after the cursor enters the bounds of the listened-to component.
- **void mouseExited(MouseEvent)**
  - Called just after the cursor exits the bounds of the listened-to component.
- **void mousePressed(MouseEvent)**
  - Called just after the user presses a mouse button while the cursor is over the listened-to component.

- **void mouseReleased(MouseEvent)**
  - Called just after the user releases a mouse button after a mouse press over the listened-to component.

The MouseEvent class defines the following methods:

int getClickCount()

- Returns the number of quick, consecutive clicks the user has made (including this event). For example, returns 2 for a double click.

- int getX()
- int getY()
- Point getPoint()
  - Return the (x,y) position at which the event occurred, relative to the component that fired the event.
- boolean isPopupTrigger()
  - Returns true if the mouse event should cause a popup menu to appear. Because popup triggers are platform dependent, if your program uses popup menus, you should call isPopupTrigger for all mouse-pressed and mouse-released events fired by components over which the popup can appear.

The MouseEvent class inherits the following method from ComponentEvent :

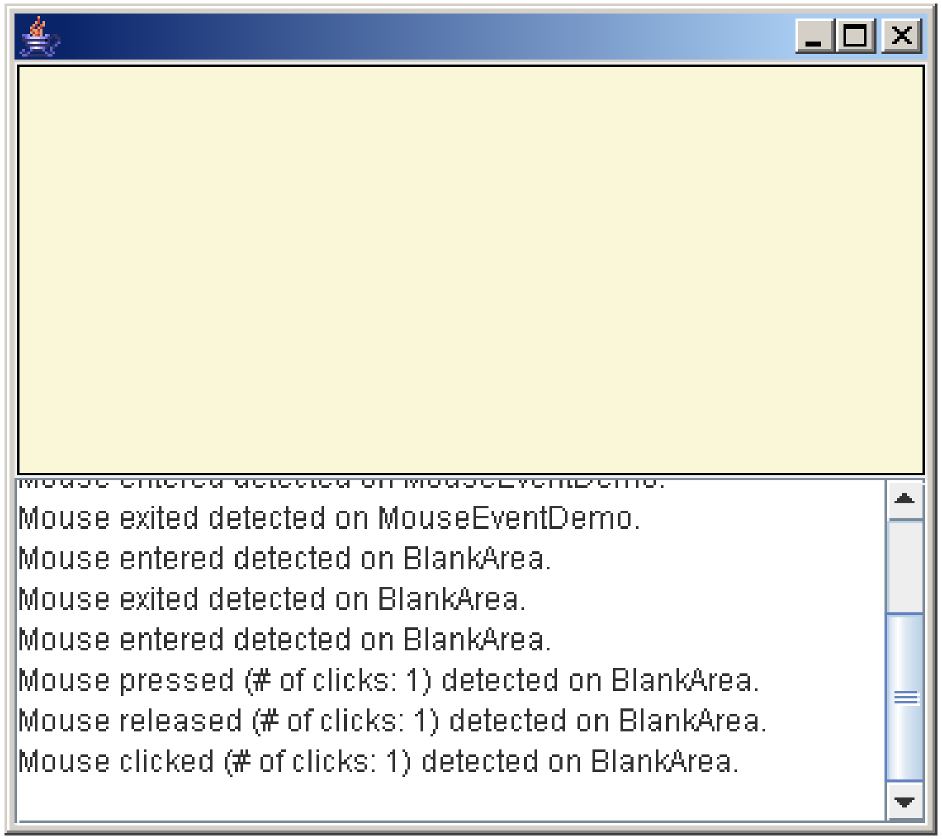
- **Component getComponent**
  - Returns the component that fired the event. You can use this method instead of the getSource method.
- The MouseEvent class inherits many useful methods from InputEvent :
- **int getWhen()**
  - Returns the timestamp of when this event occurred. The higher the timestamp, the more recently the event occurred.

The MouseEvent class inherits the following method from ComponentEvent .

- **boolean isAltDown()**
  boolean isControlDown()
  boolean isMetaDown()
  boolean isShiftDown()
  - Returns the state of individual modifier keys at the time the event was fired.

- **int getModifiers()**
  - Returns the state of all the modifier keys and mouse buttons when the event was fired. You can use this method to determine which mouse button was pressed (or newly released) when a mouse event was fired. The InputEvent class defines these constants for use with the getModifiers method: ALT_MASK, BUTTON1_MASK, BUTTON2__MASK, BUTTON3_MASK, CTRL_MASK, META_MASK, and SHIFT_MASK. For example, the following expression is true if the right button was pressed:
    (mouseEvent.getModifiers() & InputEvent.BUTTON3_MASK)
    == InputEvent.BUTTON3_MASK
  - The SwingUtilities class contains convenience methods for determining whether a particular mouse button has been pressed:
  - static boolean isLeftMouseButton(MouseEvent)
    static boolean isMiddleMouseButton(MouseEvent)
    static boolean isRightMouseButton(MouseEvent)

MouseEventDemo.java

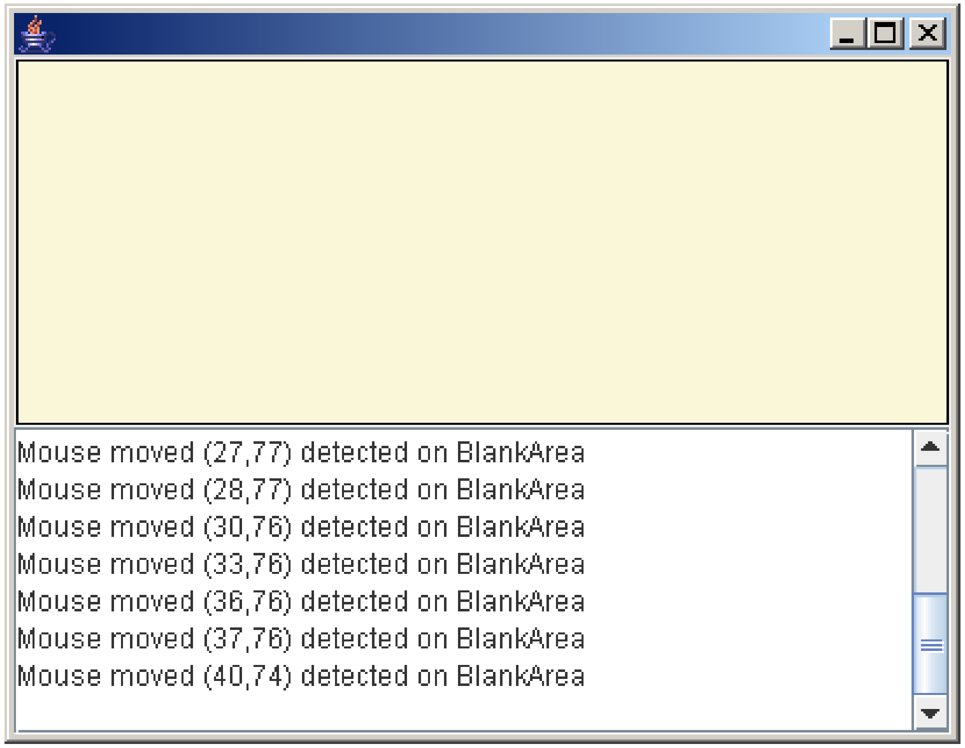


```
Mouse exited detected on MouseEventDemo.
Mouse entered detected on BlankArea.
Mouse exited detected on BlankArea.
Mouse entered detected on BlankArea.
Mouse pressed (# of clicks: 1) detected on BlankArea.
Mouse released (# of clicks: 1) detected on BlankArea.
Mouse clicked (# of clicks: 1) detected on BlankArea.
```

Mouse-Motion Listener

- Mouse-motion events tell you when the user uses the mouse (or a similar input device) to move the onscreen cursor
- The MouseMotionListener interface and adapter class, MouseMotionAdapter, contain methods:
- **void mouseDragged(MouseEvent)**
  - Called in response to the user moving the mouse while holding a mouse button down. This event is fired by the component that fired the most recent mouse-pressed event, even if the cursor is no longer over that component.
- **void mouseMoved(MouseEvent)**
  - Called in response to the user moving the mouse with no mouse buttons pressed. This event is fired by the component that's currently under the cursor.

MouseMotionEventDemo.java



```
Mouse moved (27,77) detected on BlankArea
Mouse moved (28,77) detected on BlankArea
Mouse moved (30,76) detected on BlankArea
Mouse moved (33,76) detected on BlankArea
Mouse moved (36,76) detected on BlankArea
Mouse moved (37,76) detected on BlankArea
Mouse moved (40,74) detected on BlankArea
```