

Event Driven Programming

Sean P. Strout (sps@cs.rit.edu)

Robert Duncan (rwd@cs.rit.edu)



Anonymous Classes

- An anonymous class is a local class that does not have a name.
- An anonymous class allows an object to be created using an expression that combines object creation with the declaration of the class.
- This avoids naming a class, at the cost of only ever being able to create one instance of that anonymous class.
- This is handy in the AWT (for actionListeners).

- An anonymous class is defined as part of a *new* expression and *must* be a subclass or implement an interface (without stating “extends” or “implements”).
- The class body can define methods but cannot define any constructors.

```
new className( argumentList ) { classBody }
new interfaceName() { classBody }
```

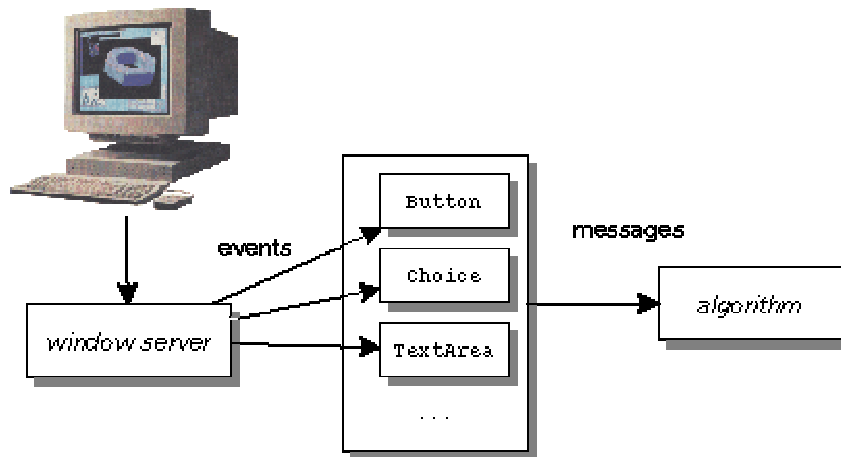
```
public class Dog {
    private String breed; private String name;


    public Dog( String theBreed, String theName ) {
        breed = theBreed; name = theName;
    }

    public String getBreed() { return breed; }
    public String getName() { return name; }

    public int compareTo( Object o ) throws ClassCastException {
        Dog other = (Dog)o;
        int retVal = breed.compareTo( other.getBreed() );
        if ( retVal == 0 )
            retVal = name.compareTo( other.getName() );
        return retVal;
    }
} // Dog
```

```
public void PrintDogsByName( List dogs ) {  
    List sorted = dogs;  
  
    Collections.sort( sorted,  
        new Comparator () {  
            public int compare( Object o1, Object o2) {  
                Dog d1 = (Dog)o1;  
                Dog d2 = (Dog)o2;  
  
                return d1.getName().compareTo( d2.getName() );  
            }  
        }  
    );  
  
    Iterator i = sorted.iterator();  
    while ( i.hasNext() )  
        System.out.println( i.next() );  
}
```






Department of
Computer Science

What is Event Driven Programming?

- Java's GUI design is based on **event driven programming**
- An **event** is a signal to the program that some external action has occurred (outside the control of the program)
 - A button was clicked
 - The mouse was moved
 - A key was pressed
 - A CD is removed from the CD drive
 - A timer in the operating system expired
- When an event is triggered, a special piece of code can run to respond to the event
 - The left mouse button was pressed so fire the current weapon
 - The mouse was moved so update the players look direction
 - The forward key was pressed so update the players position
- Event driven programming involves writing the handlers and arranging for the handler to be notified when certain events occur

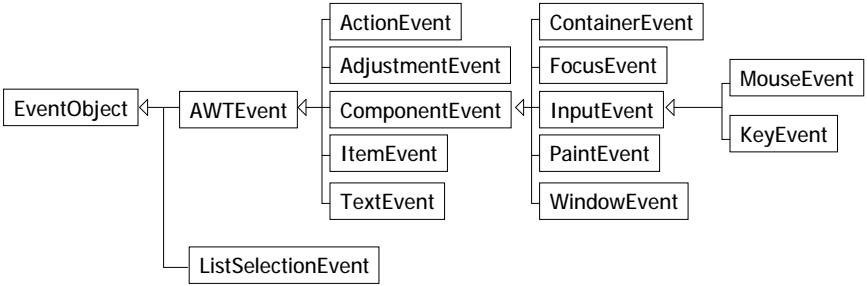
10/24/2005
Event Driven Programming
7



Department of
Computer Science

Events and Event Source

- The component which generated the event is the **source object**
 - A button is the source of a button clicking action
- The event generated is an object of `EventObject`



```

classDiagram
    class EventObject
    class AWTEvent
    class ListSelectionEvent
    class ActionEvent
    class AdjustmentEvent
    class ComponentEvent
    class ItemEvent
    class TextEvent
    class ContainerEvent
    class InputEvent
    class FocusEvent
    class MouseEvent
    class KeyEvent
    class PaintEvent
    class WindowEvent

    EventObject <|-- AWTEvent
    EventObject <|-- ListSelectionEvent
    AWTEvent <|-- ActionEvent
    AWTEvent <|-- AdjustmentEvent
    AWTEvent <|-- ComponentEvent
    AWTEvent <|-- ItemEvent
    AWTEvent <|-- TextEvent
    ComponentEvent <|-- ContainerEvent
    ComponentEvent <|-- InputEvent
    InputEvent <|-- FocusEvent
    InputEvent <|-- MouseEvent
    InputEvent <|-- KeyEvent
    InputEvent <|-- PaintEvent
    InputEvent <|-- WindowEvent
    
```

10/24/2005
Event Driven Programming
8

Events and Event Source

The diagram illustrates the relationship between GUI components and their associated actions, sources, and events. The central window, titled "GUI Components", contains several elements: an "OK" button, a text field labeled "Enter your name:" with the placeholder "Type name here", a "Bald" checkbox, a "Rad" radio button, "Green" and "Blue" radio buttons, and a "High" dropdown menu. Arrows point from text boxes to these elements, detailing their actions, sources, and events.

- Action: Key press**
Source: JTextField
Event: KeyEvent
- Action: Press return**
Source: JTextField
Event: ActionEvent
- Action: Select menu item**
Source: JMenuItem
Event: ActionEvent
- Action: Click button**
Source: JButton
Event: ActionEvent
- Action: Click box**
Source: JCheckBox
Event: ItemEvent, ActionEvent
- Action: Click button**
Source: JRadioButton
Event: ItemEvent, ActionEvent
- Action: Window iconified**
Source: Window
Event: WindowEvent

10/24/2005 Event Driven Programming 9


Listeners

- A **listener** is an object who is interested in receiving events

When this button is pressed, I want the button (the source object) to call me (the listener object) with the event and any pertinent information

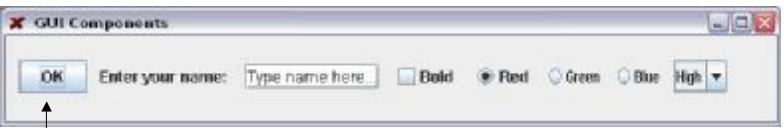
- For an object to be a listener it must do two things:
 - Implement the corresponding **event-listener interface**
 - **Register** with the source object who generates the event

10/24/2005 Event Driven Programming 10


Department of
Computer Science
Handlers


- For example, the corresponding listener interface for an `ActionEvent` is `ActionListener`
- The `ActionListener` interface requires the listener to implement the following **handler**:

```
void actionPerformed(ActionEvent event);
```




When this button is pressed, the button will call `actionPerformed()` with the `ActionEvent` to all registered listeners

10/24/2005
Event Driven Programming
11


Department of
Computer Science
Registration


- A listener **registers** with the source object by invoking a registration method in the source object
- The `JButton` object has an `addActionListener` method which takes the listener object

```
ListenerClass listener = new ListenerClass();
Jbutton button = new Jbutton("OK");
button.addActionListener(listener); ← registration
```



Every button push will cause button to call the `actionPerformed` method in listener

10/24/2005
Event Driven Programming
12


EventObject

- The **event object** contains information pertinent to the event type

```

classDiagram
    class java_util_EventObject {
        +getSource() : Object
    }
    class java_awt_event_AWTEvent {
    }
    class java_awt_event_ActionEvent {
        +getActionCommand() : String
        +getModifier() : int
        +getWhen() : long
    }
    java_util_EventObject <|-- java_awt_event_AWTEvent
    java_awt_event_AWTEvent <|-- java_awt_event_ActionEvent
    
```

java.util.EventObject

+ getSource() : Object

Returns the object on which the event initially occurred

java.awt.event.ActionEvent

+ getActionCommand() : String


+ getModifier() : int

+ getWhen() : long


Returns the command string associated with the action (i.e. the button text)

Returns the timestamp when the event occurred

10/24/2005
Event Driven Programming
13


Action Events

- Write a program that displays two buttons, OK and Cancel, in the window. A message is displayed on the console to indicate which button was pressed and when.




```


% java TestActionEvent
The OK button was clicked at
Mon Jan 24 20:13:58 EST 2005
The Cancel button was clicked at
Mon Jan 24 20:14:03 EST 2005
    
```

- `/usr/local/pub/sps/courses/cs2/events/ActionEvent`

10/24/2005
Event Driven Programming
14


 Window Events

- Write a program that demonstrates handling of window events:
 - Window opened
 - Window closing/closed
 - Window activated
 - Window deactivated
 - Window iconified
 - Window deiconified




- `/usr/local/pub/sps/courses/cs2/events/WindowEvent`

10/24/2005 Event Driven Programming 15

 WindowAdapter

- The WindowAdapter class is a class that implements the WindowListener interface
 - The methods in this class are empty.
- To use the WindowAdapter class:
 - Extend this class to create a WindowEvent listener
 - Override the methods for the events of interest
 - Create a listener object using the extended class and then register it with a Window using the window's `addWindowListener()` method.

10/24/2005 Event Driven Programming 16



Department of
Computer Science

The Result

```

import javax.swing.*;
import java.awt.event.*;


public class SwingFrame {
    public static void main( String args[] ) {
        JFrame win = new JFrame( "My First GUI Program" );

        win.addWindowListener(
            new WindowAdapter() {
                public void windowClosing( WindowEvent e ) {
                    System.exit ( 0 );
                }
            }
        );

        win.setSize( 250, 150 );
        win.setVisible(true);
    }
} // SwingFrame

```


10/24/2005
Event Driven Programming
17



Department of
Computer Science

Multiple Listeners

- Write a program which changes values by 1's and 2's using multiple listeners




```

Listener 1 created: 0
Listener 2 created: 0
Listener 2 inc: 2
Listener 1 inc: 1
Listener 2 inc: 4
Listener 1 inc: 2
Listener 2 dec: 2
Listener 1 dec: 1
Listener 2 inc: 4
Listener 1 inc: 2

```

- /usr/local/pub/sps/courses/cs2/events/MultipleListeners

10/24/2005
Event Driven Programming
18


Mouse Events

- A **mouse event** is generated whenever a mouse is pressed, released, clicked, moved or dragged on a component

java.awt.event.InputEvent


- + getWhen() : long
- + isAltDown() : boolean
- + isControlDown() : boolean
- + isMetaDown() : boolean
- + isShiftDown() : boolean

\triangle

java.awt.event.MouseEvent

- + getButton() : int ← Which mouse button was clicked?
- + getClickCount() : int ← How many times was it clicked?
- + getPoint() : java.awt.Point
- + getX() : int ← Get the coordinates for the mouse point
- + getY() : int

10/24/2005
Event Driven Programming
19


Mouse Listeners

- There are two listener interfaces to handle mouse events

java.awt.event.MouseListener


- + mousePressed(e : MouseEvent) : void
- + mouseReleased(e : MouseEvent) : void
- + mouseClicked(e : MouseEvent) : void ← Pressed and released
- + mouseEntered(e : MouseEvent) : void
- + mouseExited(e : MouseEvent) : void

←

java.awt.event.MouseMotionListener

- + mouseDragged(e : MouseEvent) : void ← Moved while pressed
- + mouseMoved(e : MouseEvent) : void


10/24/2005
Event Driven Programming
20



Department of
Computer Science


ScribbleDemo

- Write a program which uses the mouse for scribbling. You can draw with the left mouse button and erase with the right mouse button



- /usr/local/pub/sps/courses/cs2/events/ScribbleDemo

10/24/2005
Event Driven Programming
21




Department of
Computer Science

Keyboard Events

- A **key event** is generated whenever a key is pressed, released, or typed on a component

java.awt.event.InputEvent



java.awt.event.KeyEvent

+ getKeyChar() : char ← character associated with the key

+ getKeyCode() : int ← integer associated with the key


java.awt.event.KeyListener

+ keyPressed(e : KeyEvent) : void

+ keyReleased(e : KeyEvent) : void

+ keyTyped(e : KeyEvent) : void ← Key pressed + released

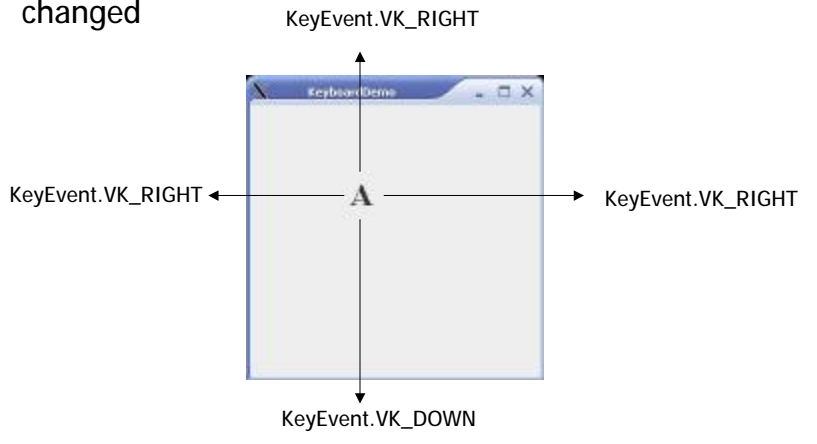
10/24/2005
Event Driven Programming
22



Department of
Computer Science


Keyboard Demo

- Write a program that displays a user input character which can be moved around with the arrow keys and changed



- /usr/local/pub/sps/courses/cs2/events/KeyboardDemo

10/24/2005
Event Driven Programming
23



Department of
Computer Science

Timers

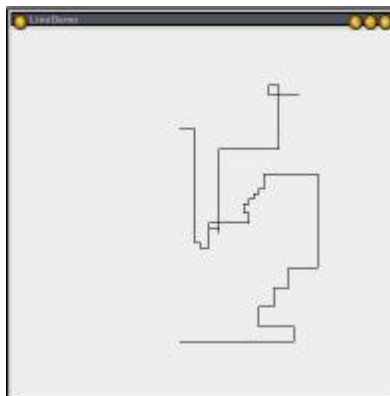
- A **timer** is a source object which can trigger an `ActionEvent` at a predefined rate
 - It's not a visible GUI component

<code>javax.swing.Timer</code>	
<code>+ Timer(delay : int, listener : ActionListener)</code>	← Create a timer with a specified delay and a listener
<code>+ addActionListener(listener : ActionListener) : void</code>	← Add a listener to the timer
<code>+ start() : void</code>	← Start, stop, or set a new delay on the timer
<code>+ stop() : void</code>	
<code>+ setDelay(delay : int) : void</code>	

10/24/2005
Event Driven Programming
24

- Write a program which modifies the keyboard demo to automatically move the character around the screen using a timer
- The character should move in the direction of the last arrow press (initial = right), every 1/10th of a second
- The character should wrap around the screen edges
- The animation should pause if the space key is pressed
- /usr/local/pub/sps/courses/cs2/events/TimerDemo

- Write a program which draws line segments using the arrow keys. The line starts from the center of the frame and draws towards east, north, west, or south when the arrow keys are pressed



- The GUI provides a view of the program, it is clearly not the program.
- Making the GUI code independent of the program code is a good strategy:
 - Changes in the program do not necessarily change the GUI.
 - Different GUIs can be developed for the same program.
 - Debugging and maintaining both the GUI and the program code can be done separately and is easier.

- The MVC pattern is commonly used to develop applications that have a GUI component
- Consists of three parts
 - Model
 - The program
 - View
 - The GUI
 - Controller
 - The event handling mechanism

