

GUI Event Handling

Typical command line program

- Non-interactive
- Linear execution

program:

```
main()
{
  code;
  code;
  code;
  code;
  code;
  code;
  code;
  code;
  code;
  code;
  code;
}
```

Interactive command line program

- User input commands
- Non-linear execution
- Unpredictable order
- Much idle time

program:

```
main()
{
  decl data storage;
  initialization code;

  loop
  {
    get command;
    switch(command)
    {
      command1:
        code;
      command2:
        code;
      ...
    }
  }
}
```

Typical GUI program

- User input commands
- Non-linear execution
- Unpredictable order
- Much idle time
- Event callback procs

GUI program:

```
main()
{
  decl data storage;
  initialization code;
  create GUI;
  register callbacks;
  main event loop;
}

Callback1() //button1 press
{
  code;
}

Callback2()
{
  code;
}
...
```

What is an Event?

- GUI components communicate with the rest of the applications through events.
- The source of an event is the component that causes that event to occur.
- The listener of an event is an object that receives the event and processes it appropriately.

Handling Events

- Every time the user types a character or clicks the mouse, an event occurs.
- Any object can be notified of any particular event.
- To be notified for an event,
 - The object has to be registered as an event listener on the appropriate event source.
 - The object has to implement the appropriate interface.

An example of Event Handling

```
public class SwingApplication implements
ActionListener {
    ...
    JButton button = new JButton("I'm a Swing
button!");
    button.addActionListener(this);
    ...
    public void actionPerformed(ActionEvent e)
    { numClicks++;
    label.setText(labelPrefix + numClicks);
    }
}
```

7

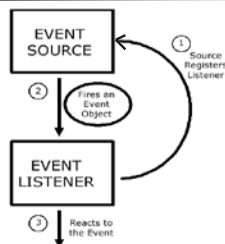
The Event Handling process

When an event is triggered, the JAVA runtime first determines its source and type. If a listener for this type of event is registered with the source, an event object is created.

For each listener to this type of an event, the JAVA runtime invokes the appropriate event handling method to the listener and passes the event object as the parameter.

8

The Event Handling Process (2)



9

What does an Event Handler require?

- It just looks for 3 pieces of code!
- First, in the declaration of the event handler class, one line of code must specify that the class implements either a listener interface or extends a class that implements a listener interface.

```
public class DemoClass implements
ActionListener {
```

10

What does an Event Handler require? (2)

- Second, it looks for a line of code which registers an instance of the event handler class as a listener of one or more components because, as mentioned earlier, the object must be registered as an event listener.

```
anyComponent.addActionListener(instance
Of DemoClass);
```

11

What does an Event Handler require? (3)

- Third, the event handler must have a piece of code that implements the methods in the listener interface.

```
public void actionPerformed(ActionEvent e)
{
    ... //code that reacts to the action ...
    // if e.source()==button1 ...
}
```

12

Types of Events

- Below, are some of the many kinds of events, swing components generate.

Act causing Event	Listener Type
User clicks a button, presses Enter, typing in text field	ActionListener
User closes a frame	WindowListener
Clicking a mouse button, while the cursor is over a component	MouseListener

13

Types of Events (2)

Act causing Event	Listener Type
User moving the mouse over a component	MouseMotionListener
Component becomes visible	ComponentListener
Table or list selection changes	ListSelectionListener

14

The Event classes

- An event object has an event class as its reference data type.
- The Event object class
 - Defined in the java.util package.
- The AWT Event class
 - An immediate subclass of EventObject.
 - Defined in java.awt package.
 - Root of all AWT based events.

15

Event Listeners

- Event listeners are the classes that implement the <type>Listener interfaces.

Example:

- ActionListener receives action events
- MouseListener receives mouse events.

The following slides give you a brief overview on some of the listener types.

16

The ActionListener Method

- It contains exactly one method.

Example:

```
public void actionPerformed(ActionEvent e)
```

The above code contains the handler for the ActionEvent e that occurred.

17

The MouseListener Methods

- Event handling when the mouse is clicked.

```
public void mouseClicked(MouseEvent e)
```
- Event handling when the mouse enters a component.

```
public void mouseEntered(MouseEvent e)
```
- Event handling when the mouse exits a component.

```
public void mouseExited(MouseEvent e)
```

18

The MouseListener Methods (2)

- Event handling when the mouse button is pressed on a component.
`public void mousePressed(MouseEvent e)`
- Event handling when the mouse button is released on a component.
`public void mouseReleased(MouseEvent e)`

19

The MouseMotionListener Methods

- Invoked when the mouse button is pressed over a component and dragged. Called several times as the mouse is dragged
`public void mouseDragged(MouseEvent e)`
- Invoked when the mouse cursor has been moved onto a component but no buttons have been pushed.
`public void mouseMoved(MouseEvent e)`

20

The WindowListener Methods

- Invoked when the window object is opened.
`public void windowOpened(WindowEvent e)`
- Invoked when the user attempts to close the window object from the object's system menu.
`public void windowClosing(WindowEvent e)`
- Invoked when the window object is closed as a result of calling dispose (release of resources used by the source).
`public void windowClosed(WindowEvent e)`

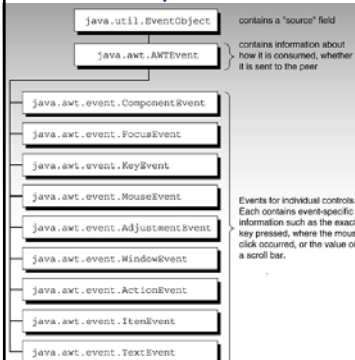
21

The WindowListener Methods (2)

- Invoked when the window is set to be the active window.
`public void windowActivated(WindowEvent e)`
- Invoked when the window object is no longer the active window
`public void windowDeactivated(WindowEvent e)`
- Invoked when the window is minimized.
`public void windowIconified(WindowEvent e)`
- Invoked when the window is changed from the minimized state to the normal state.
`public void windowDeiconified(WindowEvent e)`

22

Hierarchy of event objects



Note: The number of event objects is much greater than specified in the diagram. Only some of them are represented in the figure

Courtesy: Safari.oreilly.com
23

Additional Listener Types

- Change Listener
- Container Listener
- Document Listener
- Focus Listener
- Internal Frame Listener
- Item Listener
- Key Listener
- Property Change Listener
- Table Model Listener

The main purpose of the last few slides is to give you an idea as to how you can use event handlers in your programs. See the JAVA tutorials for more information.

24

Adapter classes for Event Handling.

- Why do you need adapter classes?
 - Implementing all the methods of an interface involves a lot of work.
 - If you are interested in only using some methods of the interface.
- Adapter classes
 - Built-in in JAVA
 - Implement all the methods of each listener interface with more than one method.
 - Implementation of all empty methods

25

Adapter classes - Illustration.

- You want create a class that implements a `MouseListener` interface, where you require only a couple of methods to be implemented. If your class directly implements the `MouseListener`, you *must* implement all five methods of this interface.
- Methods for those events you don't care about can have empty bodies

26

Illustration (2)

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

27

Illustration (3)

- What is the result?
 - The resulting collection of empty bodies can make the code harder to read and maintain.
- To help you avoid implementing empty bodies, the API generally includes an *adapter* class for each listener interface with more than one method. For example, the `MouseAdapter` class implements the `MouseListener` interface.

28

How to use an Adapter class?

```
/* Using an adapter class */
public class MyClass extends MouseAdapter {
    ....
    someObject.addMouseListener(this);
    ....
    public void mouseClicked(MouseEvent e) {
        ...//Event listener implementation goes
        // here
    }
}
```

29

Using Inner classes for Event Handling

- Consider that you want to use an adapter class but you don't want your public class to inherit from the adapter class.
- For example, you write an applet with some code to handle mouse events. As you know, JAVA does not permit multiple inheritance and hence your class cannot extend both the `Applet` and `MouseAdapter` classes.

30

Using Inner classes (contd..)

- Use a class inside your Applet subclass that extends the `MouseListener` class.

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

31

Creating GUI applications with Event Handling.

- Guidelines:
 - Create a GUI class
 - Describes the appearance of your GUI application.
 - Create a class implementing the appropriate listener interface
 - May refer to the same class as step 1.

32

Creating GUI applications with Event Handling (contd..)

- In the implementing class
 - Override all methods of the appropriate listener interface.
 - Describe in each method how you want to handle the events.
 - May give empty implementations for the methods you don't need.

33

Creating GUI applications with Event Handling (contd..)

- Register the listener object with the source
 - The object is an instantiation of the listener class specified in step 2.
 - Use the `add<Type>Listener` method.

34

Design Considerations

- The most important rule to keep in mind about event listeners is that they must execute quickly. Since all drawing and event-listening methods are executed in the same thread, a slow event listener might make the program seem unresponsive. So, consider the performance issues also when you create event handlers in your programs.

35

Design Considerations

- You can have choices on how the event listener has to be implemented. One particular solution might not fit in all situations. For example, you might choose to implement separate classes for different types of listeners. This might be a relatively easy architecture to maintain, but many classes can also result in reduced performance.

36