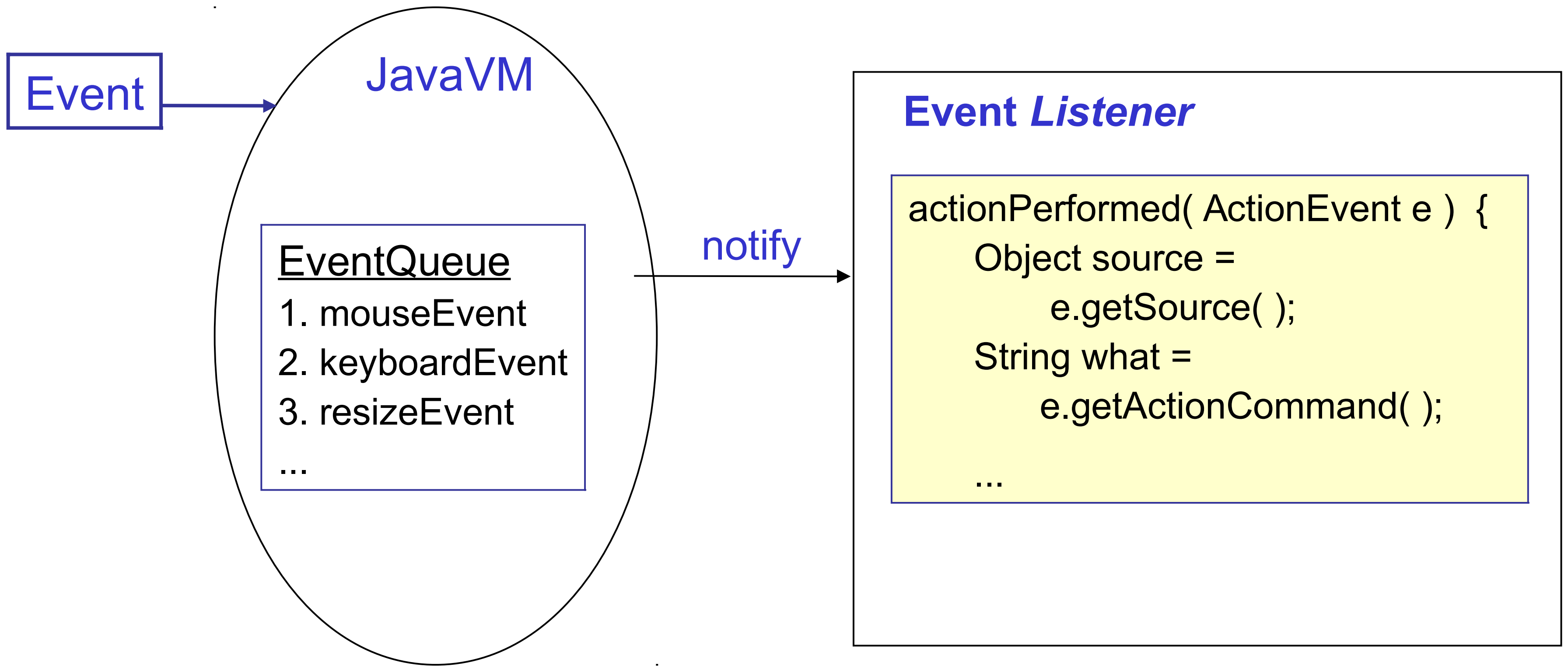
Event Handling in Swing & AWT

James Brucker

Event Driven Programming

- Graphics applications use ***events***.
- An event dispatcher receives events and ***notifies*** interested objects.



```
actionPerformed( ActionEvent e )  {
      Object source =
            e.getSource( );
      String what =
            e.getActionCommand( );
      ...
```

Step 6: Add Behavior

Your application must ***do something*** when user presses a button, moves the mouse, etc.

Graphics programs are ***event driven.***

Events:

- button press
- got focus or lost focus
- mouse movement
- text changed
- slider moved

How to Add Behavior

1) EventListeners such as **`ActionListener`**
2) **`Action`** objects - like ActionListener, but do more
3) setting properties of components

```
button.setTooltipText( "Make my day." )
```

Handling an Event

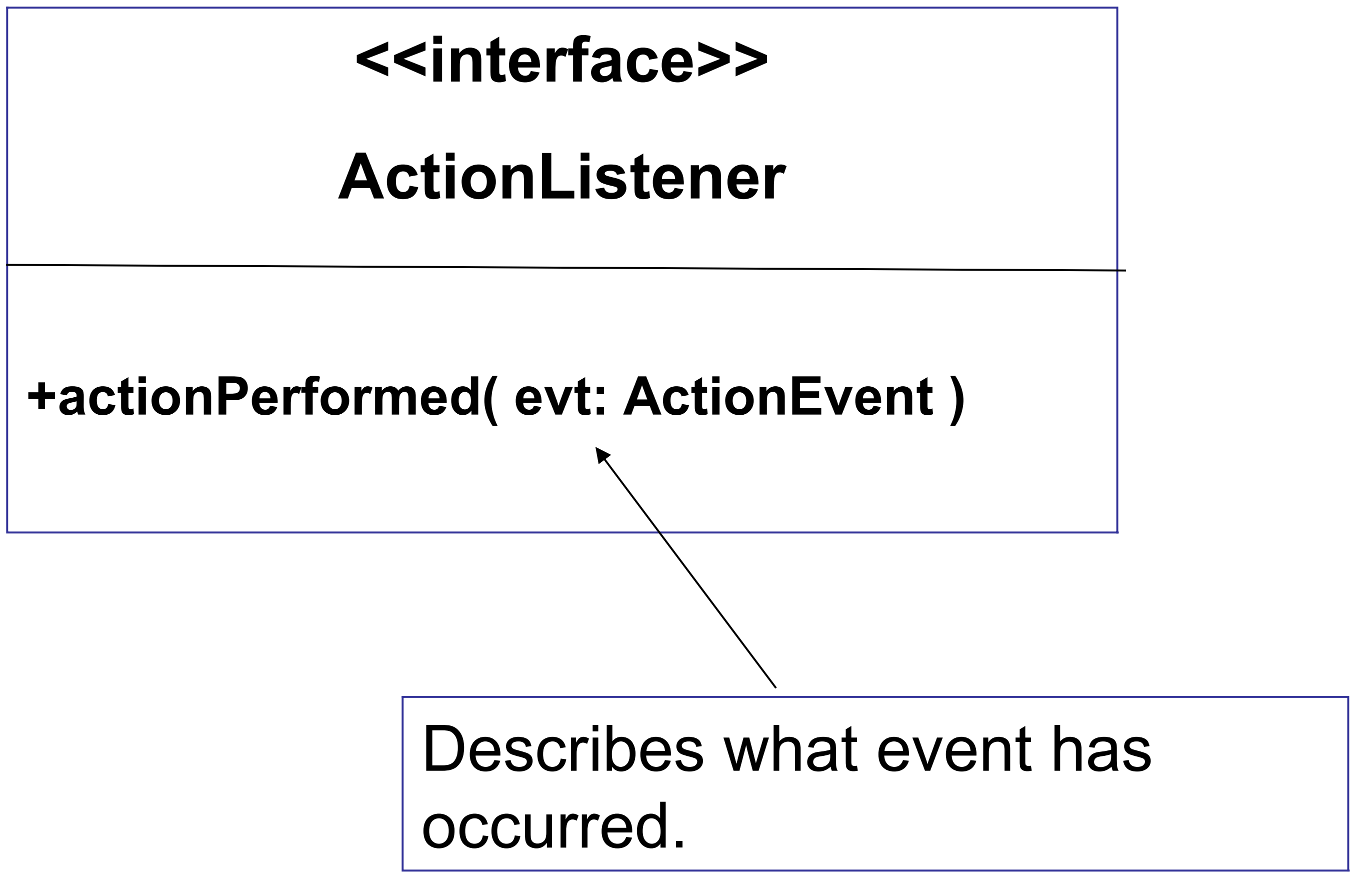
1. Write an ActionListener to handle the event.

```
ButtonListener implements ActionListener {
public void actionPerformed(ActionEvent evt) {
    do something
}
```

2. Add an instance of the Listener to a component.

```
ActionListener listener = new ButtonListener( );
button.addActionListener( listener );
```

What is an ActionListener?


<<interface>>
ActionListener
+actionPerformed(evt: ActionEvent)
Describes what event has occurred.

Write the ActionListener

- When user presses "Login" button, get username and greet him.

```
class ButtonListener implements ActionListener
{
  // actionPerformed is called when event occurs
   public void actionPerformed(ActionEvent evt) {
      String user = input.getText().trim();
      JOptionPane.showMessageDialog(frame,
           "Hello "+user);
      input.setText("");  // clear input field
   }
}
```

Add ActionListener to Login button

```
ActionListener listener =
         new ButtonListener( );

button.addActionListener( listener );
```

Event Handling Exercise

- Draw a Sequence Diagram of logic for creating and using an ActionListener.

3 Ways to Create Event Listeners

ActionListener, MouseListener, etc., are *interfaces*.

You must implement the interface the way you want.

3 Common Ways:

1. Write an (*inner) class* to implement the interface.
2. Write an *anonymous class* implement the interface.

   Anonymous class is a way to define a class without a name and create one object from the class.
3. The surrounding class implements the interface. "`this`" handles its own component events.

How To Access Private Attributes?

- The ActionListener class needs to access a private input field (input) in the GUI class:

```
    private JTextField input;
```

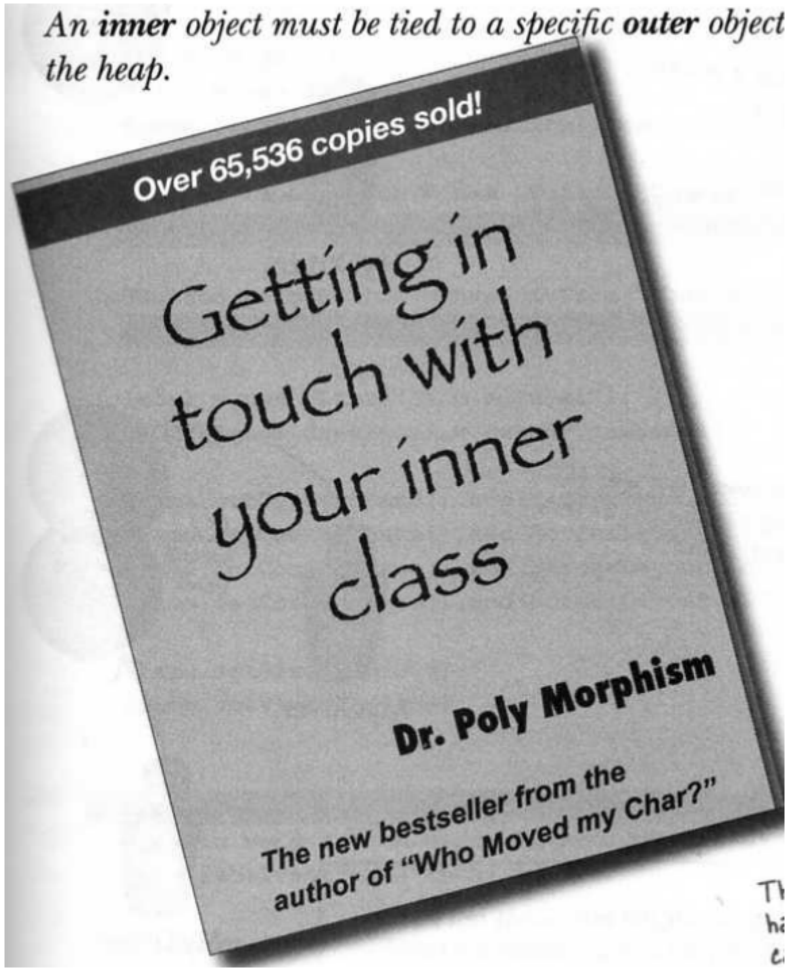
- How?

```
class ButtonListener implements ActionListener {
  public void actionPerformed(ActionEvent evt) {
      String user = input.getText().trim();
      JOptionPane.showMessageDialog(frame,
           "Hello "+user);
      input.setText("");
   }
}
```

1. Inner Class

```
public class OuterClass {
 private JTextField input;
  private JButton button;



  class ButtonListener implements ActionListener {
     public InnerClass() { /* initialize */ }
     public void actionPerformed( . . . ) {
            String text = input.getText( );
    }
 }
}
```

Inner Classes


An inner object must be tied to a specific outer object
the heap.
Over 65,536 copies sold!
Getting in touch with your inner class
Dr. Poly Morphism
The new bestseller from the author of "Who Moved my Char?"

Object of InnerClass *belongs* to an object of the OuterClass.

InnerClass object has access to all fields and methods of OuterClass object, including private ones.

InnerClass object does not exist without OuterClass object.

But: static inner class is independent of OuterClass object, just like a static method. Static inner class cannot access fields of outer class (just like a static method).

button listener using Inner Class

```
public class SwingDemo {
    private JTextField input;
   private JButton button;

   private void initCompoents() {
      button = new JButton( "Login" );
      // add an event listener to the button
      button.addActionListener(
               new ButtonHandler( ) );
      ...
   }
// a class inside the SwingDemo class
   class ButtonHandler implements ActionListener {
    public void actionPerformed(ActionEvent evt) {
      String user = input.getText().trim();
      ...
     }
   }
```

2. Anonymous Class

This code creates an *object* from an *Anonymous Class* that implements *Runnable*.

An *anonymous class* creates only one object.

```
ActionListener buttonListener =
    new ActionListener( )
   {
     public void actionPerformed(ActionEvent e){
         String text = input.getText().trim();
         . . .
     }
   };
```

Anonymous class to implement an interface.

button press using Anonymous Class

- event handler implemented as an ***Anonymous class***

```
public class SwingDemo {
...
private void initCompoents() {
   button = new JButton( "Login" );
   ActionListener buttonListener =
     new ActionListener( )
     {
        public void actionPerformed(ActionEvent evt)
        {
          String user = input.getText().trim();
           ...
        }
      };
   button.addActionListener( buttonListener );
```

Anonymous Class

button press - Outer class handles it

- you can use the application as its own listener

```
public class SwingDemo implements ActionListener {
   private JButton button;

   private void initComponents( ) {
      button = new JButton( "Press Me" );
      // add an event listener to the button
      button.addActionListener( this );
      ...

   public void actionPerformed( ActionEvent evt ) {
      String source = evt.getActionCommand( );
      label1.setText("You pressed " + source );
   }

}
```

Common Events

Action	What type of Event?	Event is handled by
Button Press	ActionEvent	ActionListener
Typing Text	TextChangedEvent	TextChangeListener
Type then press ENTER	ActionEvent	ActionListener
Mouse button press	MouseEvent	MouseListener
Mouse moved	MouseEvent	MouseMotionListener
Gain or loose focus	FocusEvent	FocusListener

How to Add an Event Listener

- You add an Event Listener to the component(s) you want events for.

Event	How to add a listener
ActionEvent	button.addActionListener(myActionListener)
TextChange Event	textbox.addTextChangeListener(myTextListener)
MouseEvent	component.addMouseListener(myMouseListener)

More Event Types

AWT Event Type	is handled by...
ActionEvent	ActionListener
AdjustmentEvent	AdjustmentListener
FocusEvent	FocusListener
ItemEvent	ItemListener
KeyEvent	KeyListener
MouseEvent	MouseListener
	MouseMotionListener
MouseWheelEvent	MouseWheelListener
WindowEvent	WindowListener
	WindowStateListener
	WindowFocusListener

Actions

An Action is an object containing an ActionListener plus *state* information.

An Action has...

- name (used as label)
- icon
- Action handler method (actionPerformed)
- key-value map
- enabled/disabled flag that enables/disables component

Action makes it *easy* to control components.

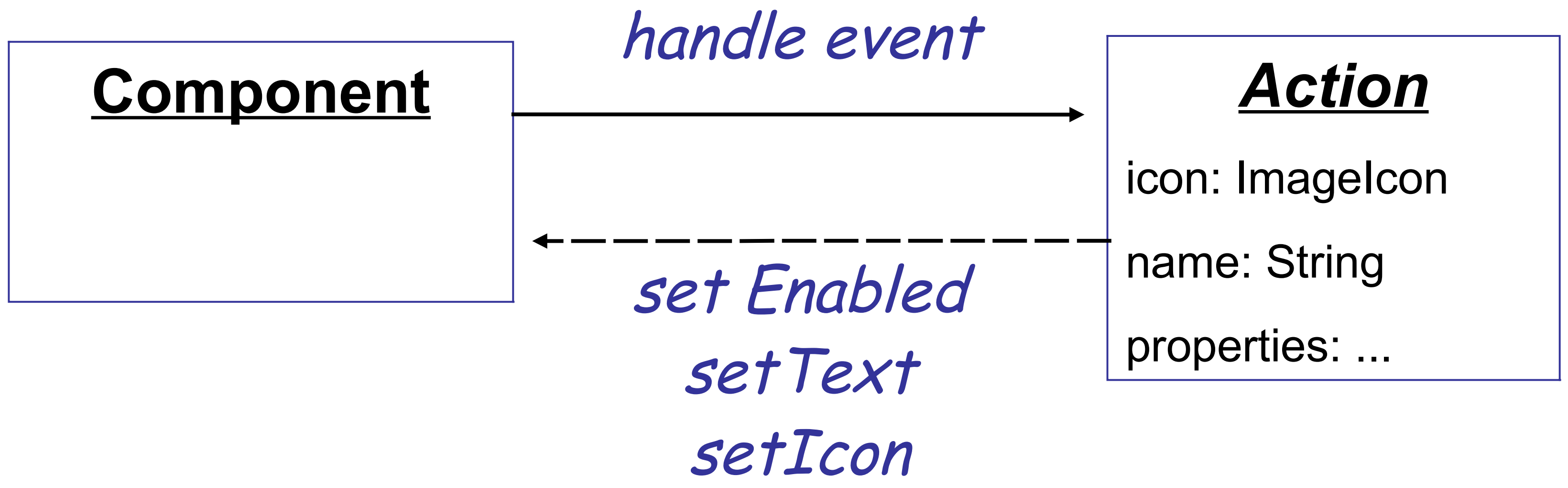
"Press Me" button using Action

```
Action loginAction = AbstractAction("Login") {
  public void actionPerformed(ActionEvent evt)
  {
      String user = input.getText().trim();
      ...
  }
}

JButton button = new JButton( loginAction );



// you can disable the button using Action...
loginAction.setEnabled(false);
```

Actions

Encapsulate behavior in an object.

Encapsulate properties.



For more info, see my slides on "Actions" or the *Java Tutorial*.