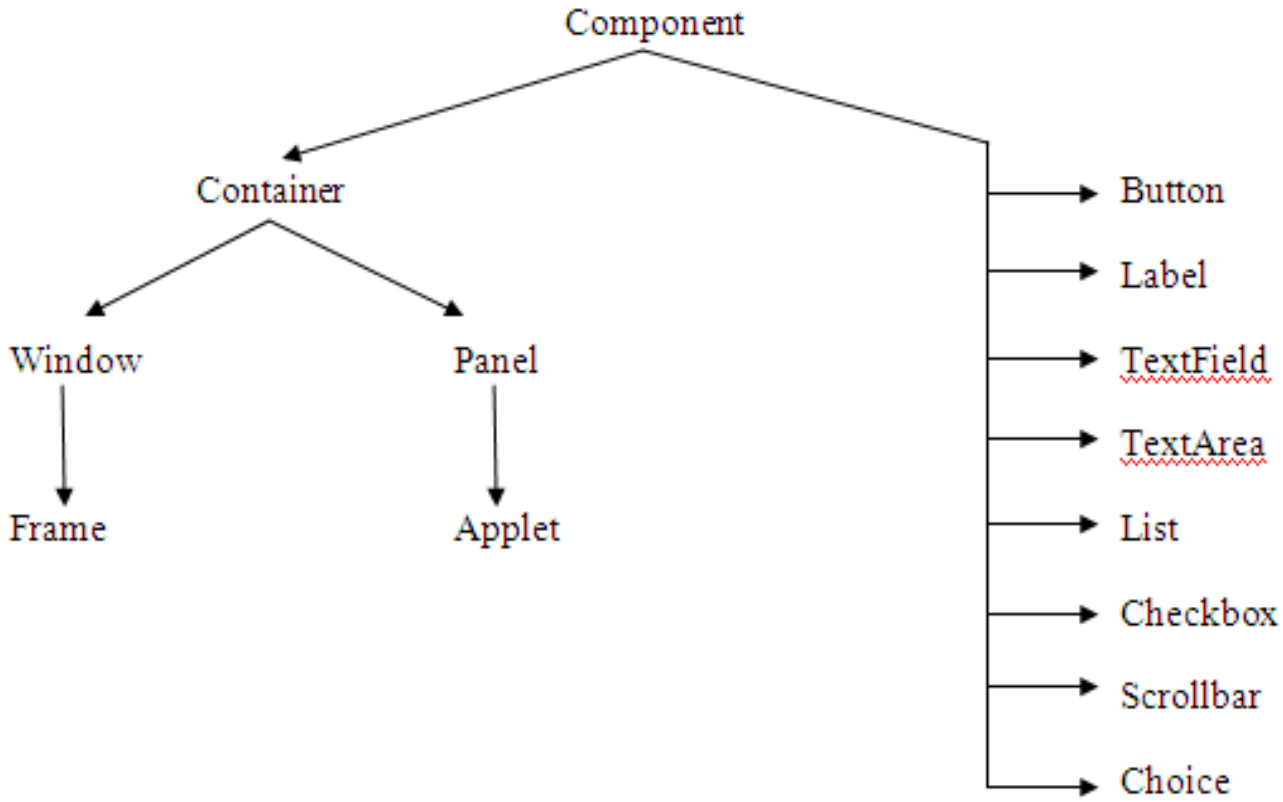
# Event Handling In Java

# Event and Listener

- Changing the state of an object is known as an event.
- For example, click on button, dragging mouse etc.
- The java.awt.event package provides many event classes and Listener interfaces for event handling.
- Any program that uses GUI (graphical user interface) such as Java application written for windows, is event driven.

# Event Handling

- To develop GUI programming in JAVA we will use awt components present in java.awt package



# Event and Listener

- Event handling has three main components,
  - **Events :** An event is a change of state of an object.
  - **Events Source :** Event source is an object that generates an event.
  - **Listeners :** A listener is an object that listens to the event. A listener gets notified when an event occurs.

- If we want to provide action to the components present in java.awt package we need to handle events generated by component.

# The Delegation Event Model

- A source generates an event and sends it to one or more listeners.
- The listener simply waits until it receives an event.
- Once an event is received, the listener processes the event and then returns.
- Application logic that processes events is separated from the user interface logic that generates those events.

# The Delegation Event Model

- A user interface element is able to "delegate" the processing of an event to a separate piece of code.
- Listeners must register with a source in order to receive an event notification.
- This provides an important benefit: notifications are sent only to listeners that want to receive them.

# Events

- An event is an object that describes a state change in a source.
- An event can be generated as a consequence of a person interacting with the elements in a GUI directly or indirectly.
- You are free to define events that are appropriate for your application.

# Event Sources

- A source is an object that generates an event.
- This occurs when the internal state of that object changes in some way. Sources may generate more than one type of event.
- A source must register listeners in order for the listeners to receive notifications about a specific type of event.
- Each type of event has its own registration method.

# Event Sources

- General Form:
  - public void addTypeListener (TypeListener el )
  - Type is the name of the event.
  - el is a reference to the event listener.

- For example, the method that registers a keyboard event listener is called addKeyListener( ).

- The method that registers a mouse motion listener is called addMouseMotionListener( ).

# Event Sources

- When an event occurs, all registered listeners are notified and receive a copy of the event object.
- This is known as multicasting the event.
- In all cases, notifications are sent only to listeners that register to receive them.
- Some sources may allow only one listener to register.
- The general form of such a method is this:
- public void addTypeListener(TypeListener el ) throws java.util.TooManyListenersException

# Event Sources

- When such an event occurs, the registered listener is notified.
- This is known as unicasting the event.

- A source must also provide a method that allows a listener to unregister an interest in a specific type of event.
- The general form of such a method is this:
- public void removeTypeListener(TypeListener el )

# Event Listeners

- A listener is an object that is notified when an event occurs.
- It has two major requirements.
- First, it must have been registered with one or more sources to receive notifications about specific types of events.
- Second, it must implement methods to receive and process these notifications.

# Event Listeners

- The methods that receive and process events are defined in a set of interfaces, such as those found in java.awt.event.
- For example, the MouseMotionListener interface defines two methods to receive notifications when the mouse is dragged or moved.
- Any object may receive and process one or both of these events if it provides an implementation of this interface.

# Event Source

| Event Source | Description |
|---|---|
| Button | Generates action events when the button is pressed. |
| Check box | Generates item events when the check box is selected or deselected. |
| Choice | Generates item events when the choice is changed. |
| List | Generates action events when an item is double-clicked; generates item events when an item is selected or deselected. |
| Menu item | Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected. |
| Scroll bar | Generates adjustment events when the scroll bar is manipulated. |
| Text components | Generates text events when the user enters a character. |
| Window | Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit. |

# Commonly Used Event Classes

| Event Class | Description |
|---|---|
| ActionEvent | Generated when a button is pressed, a list item is double-clicked, or a menu item is selected. |
| AdjustmentEvent | Generated when a scroll bar is manipulated. |
| ComponentEvent | Generated when a component is hidden, moved, resized, or becomes visible. |
| ContainerEvent | Generated when a component is added to or removed from a container. |
| FocusEvent | Generated when a component gains or loses keyboard focus. |
| InputEvent | Abstract superclass for all component input event classes. |
| ItemEvent | Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected. |
| KeyEvent | Generated when input is received from the keyboard. |
| MouseEvent | Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component. |
| MouseWheelEvent | Generated when the mouse wheel is moved. |
| TextEvent | Generated when the value of a text area or text field is changed. |
| WindowEvent | Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit. |

# Commonly Used Event Listener Interfaces

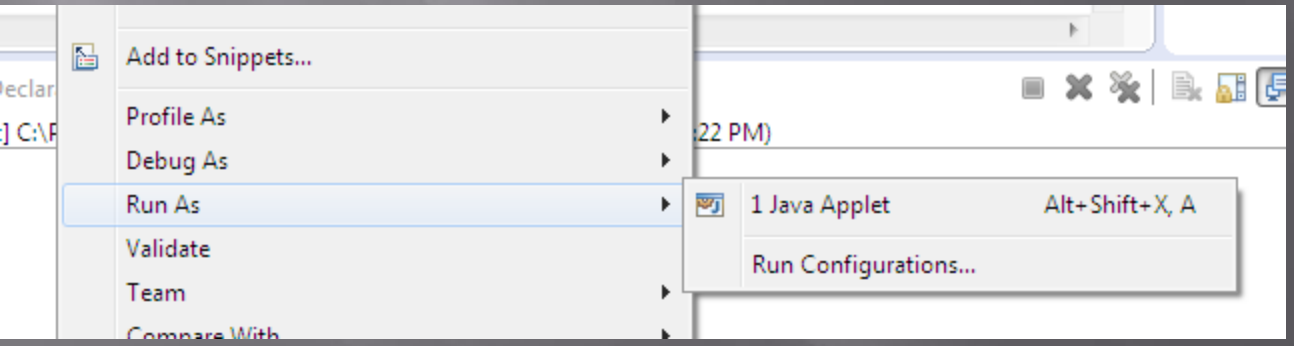
| Interface | Description |
|---|---|
| ActionListener | Defines one method to receive action events. |
| AdjustmentListener | Defines one method to receive adjustment events. |
| ComponentListener | Defines four methods to recognize when a component is hidden, moved, resized, or shown. |
| ContainerListener | Defines two methods to recognize when a component is added to or removed from a container. |
| FocusListener | Defines two methods to recognize when a component gains or loses keyboard focus. |
| ItemListener | Defines one method to recognize when the state of an item changes. |
| KeyListener | Defines three methods to recognize when a key is pressed, released, or typed. |
| MouseListener | Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released. |
| MouseMotionListener | Defines two methods to recognize when the mouse is dragged or moved. |
| MouseWheelListener | Defines one method to recognize when the mouse wheel is moved. |
| TextListener | Defines one method to recognize when a text value changes. |
| WindowFocusListener | Defines two methods to recognize when a window gains or loses input focus. |
| WindowListener | Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit. |

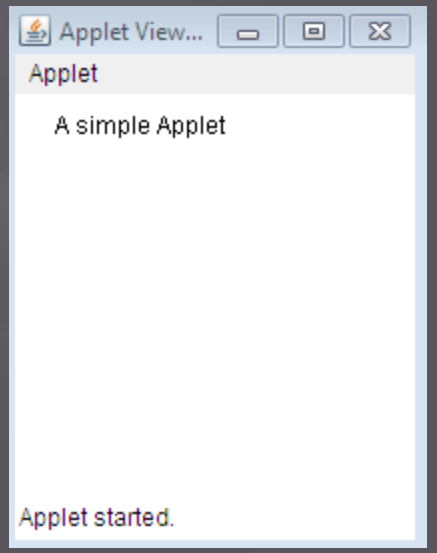
# Applet in Java

- Applets are small Java applications that can be accessed on an Internet server, transported over Internet, and can be automatically installed and run as apart of a web document. Any applet in Java is a class that extends the **java.applet.Applet** class.
- An Applet class does not have any main() method.
- It is viewed using JVM. The JVM can use either a plug-in of the Web browser or a separate runtime environment to run an applet application.
- JVM creates an instance of the applet class and invokes **init()** method to initialize an Applet.

# A Simple Applet

```
import java.awt.*;
import java.applet.*;
public class Simple extends Applet
{
 public void paint(Graphics g)
 {
  g.drawString("A simple Applet", 20, 20);
 }
}
```



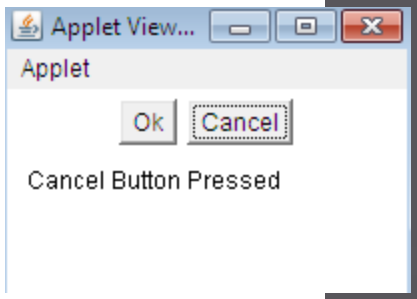



# An Example for Button Event

```
import java.applet.Applet;
import java.awt.Button;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class CreateAWTButton extends Applet implements ActionListener{
        String actionMessage="";

        public void init(){
                //create Buttons
                Button Button1 = new Button("Ok");
                Button Button2 = new Button("Cancel");
                //add Buttons
                add(Button1);
                add(Button2);
               //set action listeners for buttons
                Button1.addActionListener(this);
                Button2.addActionListener(this);
        }
        public void paint(Graphics g){
                g.drawString(actionMessage,10,50); }

        public void actionPerformed(ActionEvent ae){
                String action = ae.getActionCommand();
                if(action.equals("Ok"))
                        actionMessage = "Ok Button Pressed";
                else if(action.equals("Cancel"))
                        actionMessage = "Cancel Button Pressed";
                repaint();
        }}
```



# Example for Mouse Event Handler

```
// Demonstrate the mouse event handlers.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
  <applet code="MouseEvents" width=300 height=100>
  </applet>
*/

public class MouseEvents extends Applet
  implements MouseListener, MouseMotionListener {

  String msg = "";
  int mouseX = 0, mouseY = 0; // coordinates of mouse

  public void init() {
    addMouseListener(this);
    addMouseMotionListener(this);
  }
```

# Example for Mouse Event Handler

```
// Handle mouse clicked.
public void mouseClicked(MouseEvent me) {
   // save coordinates
   mouseX = 0;
   mouseY = 10;
   msg = "Mouse clicked.";
   repaint();
}

// Handle mouse entered.
public void mouseEntered(MouseEvent me) {
   // save coordinates
   mouseX = 0;
   mouseY = 10;
   msg = "Mouse entered.";
   repaint();
}
```

# Example for Mouse Event Handler

```
// Handle mouse exited.
public void mouseExited(MouseEvent me) {
   // save coordinates
   mouseX = 0;
   mouseY = 10;
   msg = "Mouse exited.";
   repaint();
}

// Handle button pressed.
public void mousePressed(MouseEvent me) {
   // save coordinates
   mouseX = me.getX();
   mouseY = me.getY();
   msg = "Down";
   repaint();
}

// Handle button released.
public void mouseReleased(MouseEvent me) {
   // save coordinates
   mouseX = me.getX();
   mouseY = me.getY();
   msg = "Up";
   repaint();
}
```

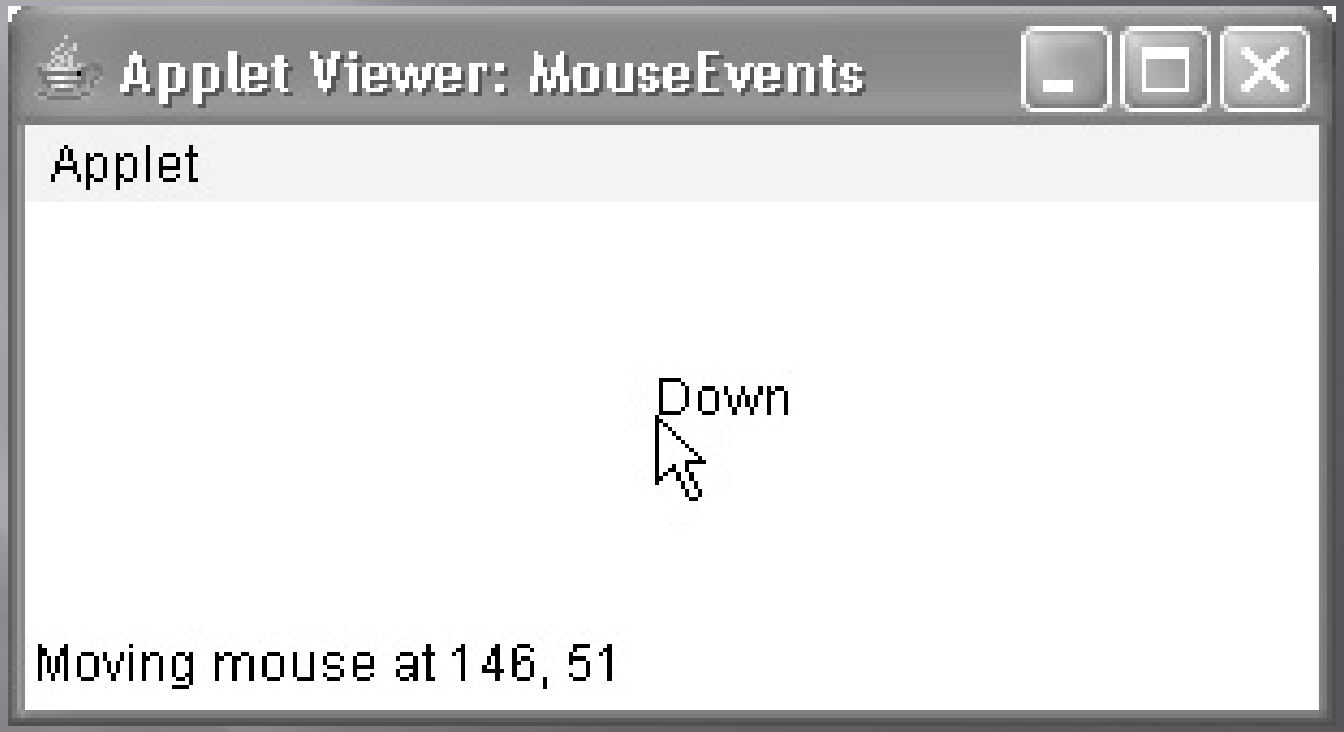
# Example for Mouse Event Handler

```
  // Handle mouse dragged.
  public void mouseDragged(MouseEvent me) {
    // save coordinates
    mouseX = me.getX();
    mouseY = me.getY();
    msg = "*";
    showStatus("Dragging mouse at " + mouseX + ", " + mouseY);
    repaint();
  }

  // Handle mouse moved.
  public void mouseMoved(MouseEvent me) {
    // show status
    showStatus("Moving mouse at " + me.getX() + ", " + me.getY());
  }

  // Display msg in applet window at current X,Y location.
  public void paint(Graphics g) {
    g.drawString(msg, mouseX, mouseY);
  }
}
```

# Example for Mouse Event Handler



# Example for Key Event Handler

```
import java.applet.*;
/*
  <applet code="SimpleKey" width=300 height=100>
  </applet>
*/

public class SimpleKey extends Applet
  implements KeyListener {

  String msg = "";
  int X = 10, Y = 20; // output coordinates

  public void init() {
    addKeyListener(this);
  }
```

# Example for Key Event Handler

```
  public void keyPressed(KeyEvent ke) {
    showStatus("Key Down");
  }

  public void keyReleased(KeyEvent ke) {
    showStatus("Key Up");
  }

  public void keyTyped(KeyEvent ke) {
    msg += ke.getKeyChar();
    repaint();
  }

  // Display keystrokes.
  public void paint(Graphics g) {
    g.drawString(msg, X, Y);
  }
}
```

# Example for Key Event Handler

# Adapter Classes

- An adapter class provides an empty implementation of all methods in an event listener interface.
- Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface.
- You can define a new class to act as an event listener by extending one of the adapter classes and implementing only those events in which you are interested.

# Adapter Classes

- For example, the MouseMotionAdapter class has two methods, mouseDragged( ) and
- mouseMoved( ), which are the methods defined by the MouseMotionListener interface.
- If you were interested in only mouse drag events, then you could simply extend MouseMotionAdapter and override mouseDragged( ).
- The empty implementation of mouseMoved( ) would handle the mouse motion events for you.

# Commonly Used Listener Interfaces Implemented by Adapter Classes

| Adapter Class | Listener Interface |
|---|---|
| ComponentAdapter | ComponentListener |
| ContainerAdapter | ContainerListener |
| FocusAdapter | FocusListener |
| KeyAdapter | KeyListener |
| MouseAdapter | MouseListener and (as of JDK 6) MouseMotionListener and MouseWheelListener |
| MouseMotionAdapter | MouseMotionListener |
| WindowAdapter | WindowListener, WindowFocusListener, and WindowStateListener |

# Adapter Class Example

```
// Demonstrate an adapter.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
  <applet code="AdapterDemo" width=300 height=100>
  </applet>
*/

public class AdapterDemo extends Applet {
  public void init() {
    addMouseListener(new MyMouseAdapter(this));
    addMouseMotionListener(new MyMouseMotionAdapter(this));
  }
}
```

# Adapter Class Example

```
class MyMouseAdapter extends MouseAdapter {

  AdapterDemo adapterDemo;
  public MyMouseAdapter(AdapterDemo adapterDemo) {
    this.adapterDemo = adapterDemo;
  }

  // Handle mouse clicked.
  public void mouseClicked(MouseEvent me) {
    adapterDemo.showStatus("Mouse clicked");
  }
}
```

# Adapter Class Example

```
class MyMouseMotionAdapter extends MouseMotionAdapter {
  AdapterDemo adapterDemo;
  public MyMouseMotionAdapter(AdapterDemo adapterDemo) {
    this.adapterDemo = adapterDemo;
  }

  // Handle mouse dragged.
  public void mouseDragged(MouseEvent me) {
    adapterDemo.showStatus("Mouse dragged");
  }
}
```

# Inner Classes, Anonymous Inner Classes

- We can handle events in an applet by using three different methods:
  - By using “this” reference
  - By using inner classes
  - By using anonymous inner classes

# Inner classes

```
// Inner class demo.
import java.applet.*;
import java.awt.event.*;
/*
  <applet code="InnerClassDemo" width=200 height=100>
  </applet>
*/

public class InnerClassDemo extends Applet {
  public void init() {
    addMouseListener(new MyMouseAdapter());
  }
  class MyMouseAdapter extends MouseAdapter {
    public void mousePressed(MouseEvent me) {
      showStatus("Mouse Pressed");
    }
  }
}
```

# Anonymous Inner Classes

```
// Anonymous inner class demo.
import java.applet.*;
import java.awt.event.*;
/*
  <applet code="AnonymousInnerClassDemo" width=200 height=100>
  </applet>
*/

public class AnonymousInnerClassDemo extends Applet {
  public void init() {
    addMouseListener(new MouseAdapter() {
      public void mousePressed(MouseEvent me) {
        showStatus("Mouse Pressed");
      }
    });
  }
}
```