Lecture 42
A W T Programming - III

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

OBJECT ORIENTED PROGRAMMING WITH JAVA

Java AWT Programming – III

Debasis Samanta
Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Concepts of Events

What is Event?

What is Event?

Events are the most significant thing in any user interface design, where users can interact with a piece of hardware or software to perform a specific operation. In fact, events add life to an interface.

An event and its consequence

Example

When we click on the "Click Me!" button an event is generated; this event, for an example, displays a frame showing a message.

An event and its consequence

Example

When we click on the "Click Me!" button an event is generated; this event, for an example, displays a frame showing a message.

Event types

Foreground events

Those events which require the direct interaction of a user. They are generated as consequences of a person interacting with the graphical components in GUIs. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from a list, scrolling the page, etc.

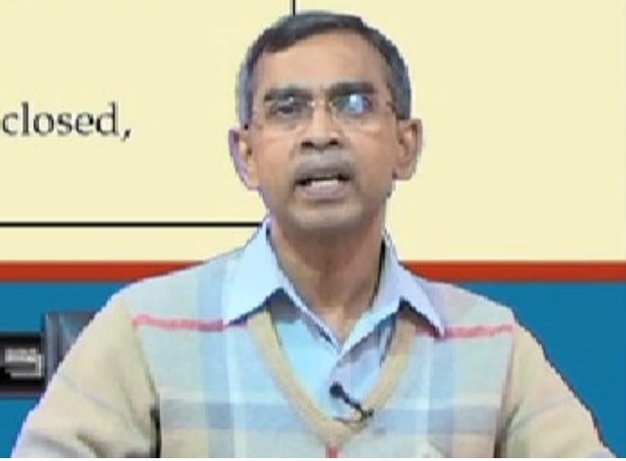
Background events

Those events that do not require any direct interaction of a user are known as background events. Operating system interrupts (due to hardware or software) are the example of background events.

Events Handling in Java

Different sources of events

Event Source	Description
Button	Generates action events when the button is pressed.
Checkbox	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.
Scrollbar	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.



Event and Component relation

Event Name	Component
ActionEvent	Button List MenuItem
ItemEvent	Checkbox Choice CheckboxMenuItem List
ComponentEvent	Component (When component is hidden, moved/resized/visible)
ContainerEvent	Container(When a component is added/removed to/from a container.)
AdjustmentEvent	Scrollbar
TextEvent	TextField (When value in a textfield is changed)
FocusEvent	When a component gain or loose Keyboard/Mouse focus.
WindowEvent	When a window is activated, closed/deactivated
KeyEvent	When an input is done from keyboard to a component
MouseEvent	When mouse is clicked/dragged/moved/released/hover a component

Event handling concept in Java

- A source generates an event and it goes to one or more listeners.
- In other words, a listener is watchful to receive an event.
- In Java, events are supported by `java.awt.event` package.
- Java provides several classes and interfaces to handle several events.

Note:

In Java, there is an abstract superclass called `InputEvent` for all component input events.

Event handling concept in Java

- A source generates an event and it goes to one or more listeners.
- In other words, a listener is watchful to receive an event.
- In Java, events are supported by `java.awt.event` package.
- Java provides several classes and interfaces to handle several events.

Note:

In Java, there is an abstract superclass called `InputEvent` for all component input events.

Event Handling Classes

Classes for event handling

Class	Description
ActionEvent	A semantic event which indicates that a component-defined action occurred.
AdjustmentEvent	The adjustment event emitted by Adjustable objects like Scrollbar and ScrollPane.
AWTEventListenerProxy	A class which extends the EventListenerProxy specifically for adding an AWTEventListener for a specific event mask.
ComponentAdapter	An abstract adapter class for receiving component events.
ComponentEvent	A low-level event which indicates that a component moved, changed size, or changed visibility (also, the root class for the other component-level events).
ContainerAdapter	An abstract adapter class for receiving container events.
ContainerEvent	A low-level event which indicates that a container's contents changed because a component was added or removed.
FocusAdapter	An abstract adapter class for receiving keyboard focus events.
FocusEvent	A low-level event which indicates that a Component has gained or lost the input focus.

Classes for event handling

Class	Description
HierarchyBoundsAdapter	An abstract adapter class for receiving ancestor moved and resized events.
HierarchyEvent	An event which indicates a change to the Component hierarchy to which Component belongs.
InputEvent	The root event class for all component-level input events.
InputMethodEvent	Input method events contain information about text that is being composed using an input method.
InvocationEvent	An event which executes the run() method on a Runnable when dispatched by the AWT event dispatcher thread.
ItemEvent	A semantic event which indicates that an item was selected or deselected.
KeyAdapter	An abstract adapter class for receiving keyboard events.
KeyEvent	An event which indicates that a keystroke occurred in a component.
MouseAdapter	An abstract adapter class for receiving mouse events.
MouseEvent	An event which indicates that a mouse action occurred in a component.
MouseMotionAdapter	An abstract adapter class for receiving mouse motion events.
MouseWheelEvent	An event which indicates that the mouse wheel was rotated in a component.
PaintEvent	The component-level paint event.
TextEvent	A semantic event which indicates that an object's text changed.
WindowAdapter	An abstract adapter class for receiving window events.
WindowEvent	A low-level event that indicates that a window has changed its status.

Classes for event handling : ActionEvent

An **ActionEvent** is generated when a button is pressed, a list item is double-clicked, or a menu item is selected. The **ActionEvent** class defines four integer constants that can be used to identify any modifiers associated with an action event:

`ALT_MASK, CTRL_MASK, META_MASK, and SHIFT_MASK.`

In addition, there is an integer constant, `ACTION_PERFORMED`, which can be used to identify action events

Constructors
`ActionEvent(Object src, int type, String cmd)`
`ActionEvent(Object src, int type, String cmd, int modifiers)`
`ActionEvent(Object src, int type, String cmd, long when, int modifiers)`

Here, `src` is a reference to the object that generated this event. The `type` of the event is specified by type, and its command string is `cmd`. The argument `modifiers` indicates which modifier keys (ALT, CTRL, META, and/or SHIFT) were pressed when the event was generated. The `when` parameter specifies when the event occurred. The third constructor was added by Java 2, version 1.4

Classes for event handling : AdjustmentEvent

An **AdjustmentEvent** is generated by a *scroll bar*. There are five types of adjustment events. The **AdjustmentEvent** class defines integer constants that can be used to identify them. The constants and their meanings are shown here:

Constants	Definition
BLOCK_DECREMENT	The user clicked inside the scroll bar to decrease its value.
BLOCK_INCREMENT	The user clicked inside the scroll bar to increase its value.
TRACK	The slider was dragged
UNIT_DECREMENT	The button at the end of the scroll bar was clicked to decrease its value.
UNIT_INCREMENT	The button at the end of the scroll bar was clicked to increase its value.

Constructor
AdjustmentEvent(Adjustable src, int id, int type, int data)

Methods	Definition
getAdjustable( )	It returns the object that generated the event.
int getValue( )	The amount of the adjustment can be obtained.

Classes for event handling : ComponentEvent

A **ComponentEvent** is generated when the size, position, or visibility of a component is changed. There are four types of component events. The **ComponentEvent** class defines integer constants that can be used to identify them.

Constants	Definition
COMPONENT_HIDDEN	The component was hidden.
COMPONENT_MOVED	The component was moved.
COMPONENT_RESIZED	The component was resized.
COMPONENT_SHOWN	The component became visible.

Constructor
ComponentEvent(Component src, int type)

Methods	Definition
getComponent( )	It returns the component that generated the event.

Classes for event handling : ContainerEvent

A **ContainerEvent** is generated when a component is added to or removed from a container. There are two types of container events.

Constants	Definition
COMPONENT_ADDED	The component was added.
COMPONENT_REMOVED	The component was removed.

Constructor
ContainerEvent(Component src, int type, Component comp)

Methods	Definition
getChild( )	returns a reference to the component that was added to or removed from the container
getContainer( )	Obtain a reference to the container that [illegible]d this event

Classes for event handling : FocusEvent

A **FocusEvent** is generated when a component gains or loses input focus.

Constants	Definition
FOCUS_GAINED	The component has been focused.
FOCUS_LOST	The component lost its focus.

Constructors
FocusEvent(Component src, int type)
FocusEvent(Component src, int type, boolean temporaryFlag)
Focus Event(Component src, int type, boolean temporaryFlag, Component other)

Methods	Definition
getOppositeComponent( )	To determine the other component
isTemporary( )	It indicates if this focus change is temporary

Classes for event handling : InputEvent

The abstract class **InputEvent** is a subclass of **ComponentEvent** and is the superclass for component input events. Its subclasses are **KeyEvent** and **MouseEvent**.

Constants
ALT_MASK
ALT_GRAPH_MASK
BUTTON1_MASK
BUTTON2_MASK
BUTTON3_MASK
CTRL_MASK
META_MASK
SHIFT_MASK

Methods	Definition
int getModifiers()	To obtain a value that contains all of the original modifier flags

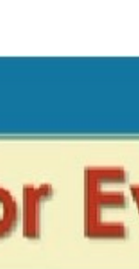
Classes for Event Handling : ItemEvent

An **ItemEvent** is generated when a check box or a list item is clicked or when a checkable menu item is selected or deselected.

Constants	Definition
DESELECTED	The user deselected an item.
SELECTED	The user selected an item.

Constructor
ItemEvent(ItemSelectable src, int type, Object entry, int state)

Methods	Definition
getItem( )	It can be used to obtain a reference to the item that generated an event.
getItemSelectable( )	It can be used to obtain a reference to the ItemSelectable object that generated an event
getStateChange( )	It returns the state change (i.e., SELECTED or DESELECTED) for the event.



DEBASIS SAMANTA
IIT KHARAGPUR


Classes for Event Handling : KeyEvent

A **KeyEvent** is generated when keyboard input occurs.

Constants	Definition
KEY_PRESSED	This event is generated when any key is pressed
KEY_RELEASED	This event is generated when any key is released
KEY_TYPED	This event is generated when a character is generated

Constructor
KeyEvent(Component src, int type, long when, int modifiers, int code, char ch)
KeyEvent(Component src, int type, long when, int modifiers, int code)

Methods	Definition
char getKeyChar( )	It returns the character that was entered.
int getKeyCode( )	It returns the key code.

Classes for event handling : MouseEvent

A `MouseEvent` is generated when mouse input occurs

Constants	Definition
MOUSE_CLICKED	The user clicked the mouse.
MOUSE_DRAGGED	The user dragged the mouse
MOUSE_ENTERED	The mouse entered a component.
MOUSE_EXITED	The mouse exited from a component.
MOUSE_MOVED	The mouse moved.
MOUSE_PRESSED	The mouse was pressed.
MOUSE_RELEASED	The mouse was released.
MOUSE_WHEEL	The mouse wheel was moved (Java 2, v1.4)

Methods	Definition
`char getKeyChar( )`	It returns the character that was entered.
`int getKeyCode( )`	It returns the key code.

Constructor

```
MouseEvent(Component src, int type, long when, int modifiers,
    int x, int y, int clicks, boolean triggersPopup)
```


Classes for Event Handling : TextEvent

Instances of **TextEvent** class describe text events. These are generated by text fields and text areas when characters are entered by a user or program.

Constants	Definition
TEXT_VALUE_CHANGED	When an update in the text is triggered

Constructor
TextEvent(Object src, int type)

Classes for Event Handling : WindowEvent

A **WindowEvent** is generated when window container get some changes.

Constants	Definition
WINDOW_ACTIVATED	This event is generated when any key is pressed
WINDOW_CLOSED	This event is generated when any key is released
WINDOW_CLOSING	This event is generated when a character is generated
WINDOW_DEACTIVATED	The window was deactivated.
WINDOW_DEICONIFIED	The window was deiconified.
WINDOW_GAINED_FOCUS	The window gained input focus
WINDOW_ICONIFIED	The window was iconified.
WINDOW_LOST_FOCUS	The window lost input focus.
WINDOW_OPENED	The window was opened.
WINDOW_STATE_CHANGED	The state of the window changed.

Constructor
`WindowEvent(Window src, int type, Window other)`
`WindowEvent(Window src, int type, int fromState, int toState)`
`WindowEvent(Window src, int type, Window other, int fromState, int toState)`

Methods	Definition
`getWindow( )`	It returns the Window object that generated the event
`getOppositeWindow()`	It returns the opposite Window object
`getOldState()`	It returns the details before modification
`getNewState()`	It returns the modification details

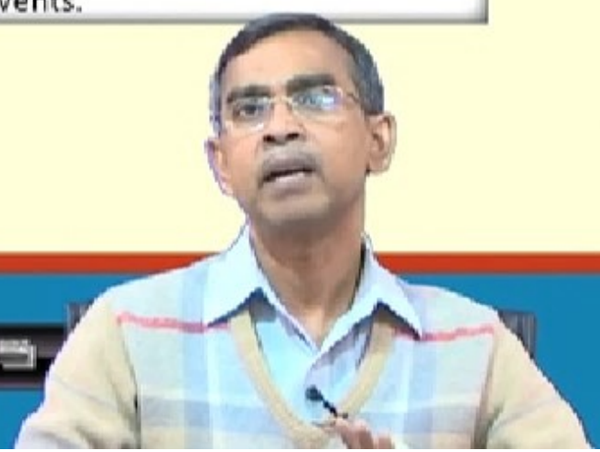
Event Handling Interfaces

Interfaces for event handling

Interface	Description
ActionListener	The listener interface for receiving action events.
AdjustmentListener	The listener interface for receiving adjustment events.
AWTEventListener	The listener interface for receiving notification of events dispatched to objects that are instances of Component or MenuComponent or their subclasses.
ComponentListener	The listener interface for receiving component events.
ContainerListener	The listener interface for receiving container events.
FocusListener	The listener interface for receiving keyboard focus events on a component.
HierarchyBoundsListener	The listener interface for receiving ancestor moved and resized events.
HierarchyListener	The listener interface for receiving hierarchy changed events.
InputMethodListener	The listener interface for receiving input method events.

Interfaces for event handling

Interface	Description
ItemListener	The listener interface for receiving item events.
KeyListener	The listener interface for receiving keyboard events (keystrokes).
MouseListener	The listener interface for receiving "interesting" mouse events (press, release, click, enter, and exit) on a component.
MouseMotionListener	The listener interface for receiving mouse motion events on a component.
MouseWheelListener	The listener interface for receiving mouse wheel events on a component.
TextListener	The listener interface for receiving text events.
WindowFocusListener	The listener interface for receiving WindowEvents, including WINDOW_GAINED_FOCUS and WINDOW_LOST_FOCUS events.
WindowListener	The listener interface for receiving window events.
WindowStateListener	The listener interface for receiving window state events.



Interfaces for event handling : ActionListener

This interface defines the `actionPerformed( )` method that is invoked when an action event occurs

Methods
`void actionPerformed(ActionEvent ae)`

Interfaces for Event Handling : AdjustmentListener

This interface defines the adjustment `ValueChanged( )` method that is invoked when an adjustment event occurs.

Methods
`void adjustmentValueChanged(AdjustmentEvent ae)`

Interfaces for event handling : ComponentListener

This interface defines four methods that are invoked when a component is resized, moved, shown, or hidden

Methods
`void componentResized(ComponentEvent ce)`
`void componentMoved(ComponentEvent ce)`
`void componentShown(ComponentEvent ce)`
`void componentHidden(ComponentEvent ce)`

Note: The AWT processes the resize and move events. The componentResized( ) and componentMoved( ) methods are provided for notification purposes only.

Interfaces for Event Handling : ContainerListener

This interface contains two methods. When a component is added to a container, componentAdded( ) is invoked. When a component is removed from a container, componentRemoved( ) is invoked.

Methods
void componentAdded(ContainerEvent ce)
void componentRemoved(ContainerEvent ce)

Interfaces for event handling : FocusListener

This interface defines two methods. When a component obtains keyboard focus, focusGained( ) is invoked. When a component loses keyboard focus, focusLost( ) is called.

Methods
void focusGained(FocusEvent fe)
void focusLost(FocusEvent fe)

Interfaces for event handling : KeyListener

This interface defines three methods. The keyPressed( ) and keyReleased( ) methods are invoked when a key is pressed and released, respectively. The keyTyped( ) method is invoked when a character has been entered.

Methods
void keyPressed(KeyEvent ke)
void keyReleased(KeyEvent ke)
void keyTyped(KeyEvent ke)

Interfaces for event handling : KeyListener

This interface defines five methods. If the mouse is pressed and released at the same point, mouseClicked( ) is invoked. When the mouse enters a component, the mouseEntered( ) method is called. When it leaves, mouseExited( ) is called. The mousePressed( ) and mouseReleased( ) methods are invoked when the mouse is pressed and released, respectively.

Methods
`void mouseClicked(MouseEvent me)`
`void mouseEntered(MouseEvent me)`
`void mouseExited(MouseEvent me)`
`void mousePressed(MouseEvent me)`
`void mouseReleased(MouseEvent me)`

Interfaces for event handling : MouseMotionListener

This interface defines two methods. The mouseDragged( ) method is called multiple times as the mouse is dragged. The mouseMoved( ) method is called multiple times as the mouse is moved

Methods
void mouseDragged(MouseEvent me)
void mouseMoved(MouseEvent me)

Interfaces for event handling : MouseWheelListener

This interface defines the mouse WheelMoved( ) method that is invoked when the mouse wheel is moved.

Methods
`void mouseWheelMoved(MouseWheelEvent mwe)`

Interfaces for Event Handling : TextListener

This interface defines the `textChanged( )` method that is invoked when a change occurs in a text area or text field.

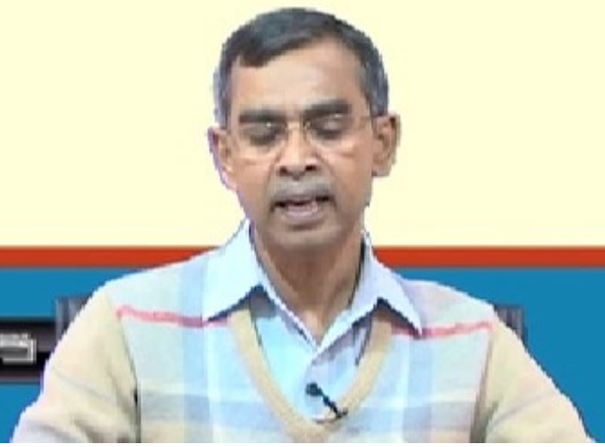
Methods
`void textChanged(TextEvent te)`

Interfaces for event handling : WindowFocusListener

This interface defines two methods: windowGainedFocus( ) and windowLostFocus( ). These are called when a window gains or losses input focus.

Methods
`void windowGainedFocus(WindowEvent we)`
`void windowLostFocus(WindowEvent we)`

Note: `WindowFocusListener` was added by Java 2, version 1.4.



Interfaces for Event Handling : WindowListener

This interface defines seven methods. The windowActivated( ) and windowDeactivated( ) methods are invoked when a window is activated or deactivated, respectively. If a window is iconified, the windowIconified( ) method is called. When a window is deiconified, the windowDeiconified( ) method is called. When a window is opened or closed, the windowOpened( ) or windowClosed( ) methods are called, respectively. The windowClosing( ) method is called when a window is being closed.

Methods
void windowActivated(WindowEvent we)
void windowClosed(WindowEvent we)
void windowClosing(WindowEvent we)
void windowDeactivated(WindowEvent we)
void windowDeiconified(WindowEvent we)
void windowIconified(WindowEvent we)
void windowOpened(WindowEvent we)

Event Handling Model in Java

Event handling mechanism

1. **Register** the source(s) to receive notifications about specific type of events.

   Each type of event has its own registration method. In general,

   ```
   public void add<TypeListner>(<TypeLister l>)
   ```

2. **Implement** the listener method(s) to receive and process these notifications.

Steps involved in event handling

1. User interacts a source and then the corresponding event is generated.
    - Now the object of concerned event class is created automatically and information about the source and the event get populated within the object.

2. Event object is forwarded to the method of registered listener class.
    - The method is now get executed and returns.

Handling Mouse Events

Familiarize with the MouseEvent class

There are eight types of mouse events. The `MouseEvent` class defines the following integer constants that can be used to identify them:

Mouse events	Description
MOUSE_CLICKED	The user clicked the mouse.
MOUSE_DRAGGED	The user dragged the mouse.
MOUSE_ENTERED	The mouse entered a component.
MOUSE_EXITED	The mouse exited from a component.
MOUSE_MOVED	The mouse moved.
MOUSE_PRESSED	The mouse was pressed.
MOUSE_RELEASED	The mouse was released.
MOUSE_WHEEL	The mouse wheel was moved (Java 2, v1.4).

Familiarize with the MouseEvent class

There are eight types of mouse events. The MouseEvent class defines the f

Note:

MouseEvent class is a subclass of InputEvent. Here, is one of its constructors.

```
MouseEvent(Component src,
           int type,
           long when,
           int modifiers,
           int x,
           int y,
           int clicks,
           boolean triggersPopup)
```

Class MouseEvent : Methods

Methods	Description
`getClickCount()`	Return the number of mouse clicks associated with this event.
`getPoint()`	Returns the x,y position of the event relative to the source component.
`getX()`	Returns the horizontal x position of the event relative to the source component.
`getY()`	Returns the vertical y position of the event relative to the source component.
`isPopupTrigger()`	Returns whether or not this mouse event is the popup-menu trigger event for the platform.
`paramString()`	Returns a parameter string identifying this event.
`translatePoint(int x,int y)`	Translates the event's coordinates to a new position by adding specified x (hori y (veritcal) offsets.

Class MouseEvent : Methods

Methods	Description
`getClickCount()`	Return the number of mouse clicks associated with this event.
`getPoint()`	Returns the x,y position of the event relative to the source component.
`getX()`	Returns the horizontal x position of the event relative to the source component.
`getY()`	Returns the vertical y position of the event relative to the source component.
`isPopupTrigger()`	Returns whether or not this mouse event is the popup-menu trigger event for the platform.
`paramString()`	Returns a parameter string identifying this event.
`translatePoint(int x,int y)`	Translates the event's coordinates to a new position by adding specified x (horizontal) and y (veritcal) offsets.

Interfaces dealing with mouse events

There are two interfaces for the purpose :

- `MouseListner` interface
- `MouseMotionListner` interface

Interface MouseListner : Methods

Methods	Description
`void mouseClicked(MouseEvent me)`	This method is called whenever the mouse button is pressed and released at the same time.
`void mouseEntered(MouseEvent me)`	When a mouse pointer enters a component.
`void mouseExited(MouseEvent me)`	When a mouse pointer exits a component.
`void mousePressed(MouseEvent me)`	This method is called whenever the mouse button is pressed
`void mouseReleased(MouseEvent me)`	This method is called whenever the mouse button is released

Interface MouseMotionListner : Methods

Methods	Description
`void mouseDragged(MouseEvent me)`	This method is called multiple times as the mouse is dragged(Clicked and Moved).
`void mouseMoved(MouseEvent me)`	This method is called multiple times as the mouse is moved (without clicking)

Implementation: Mouse event handling

1. Do the following in the `init()` method of your applet class.

```
add MouseListner(this);
add MouseMotionListner(this);
```

2. Then implement the interface methods as per your requirement. Go on adding all these methods just after `init()` method.

That's all!

Scribble 1 applet : An example

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class Scribble1 extends Applet implements MouseListener,
MouseMotionListener{
    private int last_x, last_y;

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

    public void mousePressed(MouseEvent e) {
        last_x = e.getX();
        last_y = e.getY();
    }
    public void mouseDragged(MouseEvent e) {
        Graphics g = this.getGraphics();
        int x = e.getX(), y = e.getY();
        g.drawLine(last_x, last_y, x, y);
        last_x = x; lst_y = y;
    }
// … Contd. to next
```

```
// Other methods …

    public void mouseClicked(MouseEvent e) { }
    public void mouseEntered(MouseEvent e) { }
    public void mouseExited(MouseEvent e)  { }
    public void mousePressed(MouseEvent e) { }
    public void mouseReleased(MouseEvent e){ }
}
```

Scribble 1 applet : An example

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class Scribble1 extends Applet implements MouseListener,
MouseMotionListener{
    private int last_x, last_y;

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

    public void mousePressed(MouseEvent e) {
        last_x = e.getX();
        last_y = e.getY();
    }
    public void mouseDragged(MouseEvent e) {
        Graphics g = this.getGraphics();
        int x = e.getX(), y = e.getY();
        g.drawLine(last_x, last_y, x, y);
        last_x = x; lst_y = y;
    }
// … Contd. to next
```

```
// Other methods …

    public void mouseClicked(MouseEvent e) { }
    public void mouseEntered(MouseEvent e) { }
    public void mouseExited(MouseEvent e)  { }
    public void mousePressed(MouseEvent e) { }
    public void mouseReleased(MouseEvent e){ }
}
```

Scribble 1 applet : An example

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class Scribble1 extends Applet implements MouseListener,
MouseMotionListener{
    private int last_x, last_y;

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

   public void mousePressed(MouseEvent e) {
        last_x = e.getX();
        last_y = e.getY();
   }
   public void mouseDragged(MouseEvent e) {
        Graphics g = this.getGraphics();
        int x = e.getX(), y = e.getY();
        g.drawLine(last_x, last_y, x, y);
        last_x = x; lst_y = y;
   }
// … Contd. to next
```

```
// Other methods …

    public void mouseClicked(MouseEvent e) { }
    public void mouseEntered(MouseEvent e) { }
    public void mouseExited(MouseEvent e)  { }
    public void mousePressed(MouseEvent e) { }
    public void mouseReleased(MouseEvent e){ }
}
```


Scribble 1 applet : An example

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class Scribble1 extends Applet implements MouseListener,
MouseMotionListener{
    private int last_x, last_y;

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

   public void mousePressed(MouseEvent e) {
        last_x = e.getX();
        last_y = e.getY();
   }
   public void mouseDragged(MouseEvent e) {
        Graphics g = this.getGraphics();
        int x = e.getX(), y = e.getY();
        g.drawLine(last_x, last_y, x, y);
        last_x = x; lst_y = y;
   }
// … Contd. to next
```

```
// Other methods …

    public void mouseClicked(MouseEvent e) { }
    public void mouseEntered(MouseEvent e) { }
    public void mouseExited(MouseEvent e)  { }
    public void mousePressed(MouseEvent e) { }
    public void mouseReleased(MouseEvent e){ }
}
```

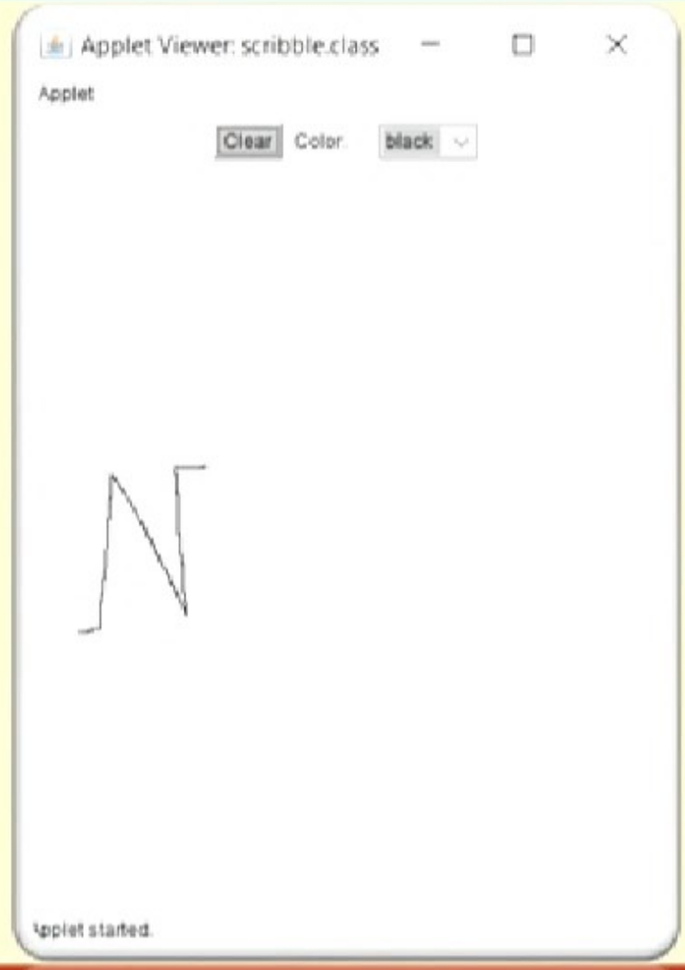
Scribble 1 applet : An example

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class Scribble1 extends Applet implements MouseListener,
MouseMotionListener{
    private int last_x, last_y;

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

   public void mousePressed(MouseEvent e) {
        last_x = e.getX();
        last_y = e.getY();
   }
   public void mouseDragged(MouseEvent e) {
        Graphics g = this.getGraphics();
        int x = e.getX(), y = e.getY();
        g.drawLine(last_x, last_y, x, y);
        last_x = x; lst_y = y;
   }
// … Contd. to next
```

```
// Other methods …

    public void mouseClicked(MouseEvent e) { }
    public void mouseEntered(MouseEvent e) { }
    public void mouseExited(MouseEvent e)  { }
    public void mouseReleased(MouseEvent e){ }
}
```

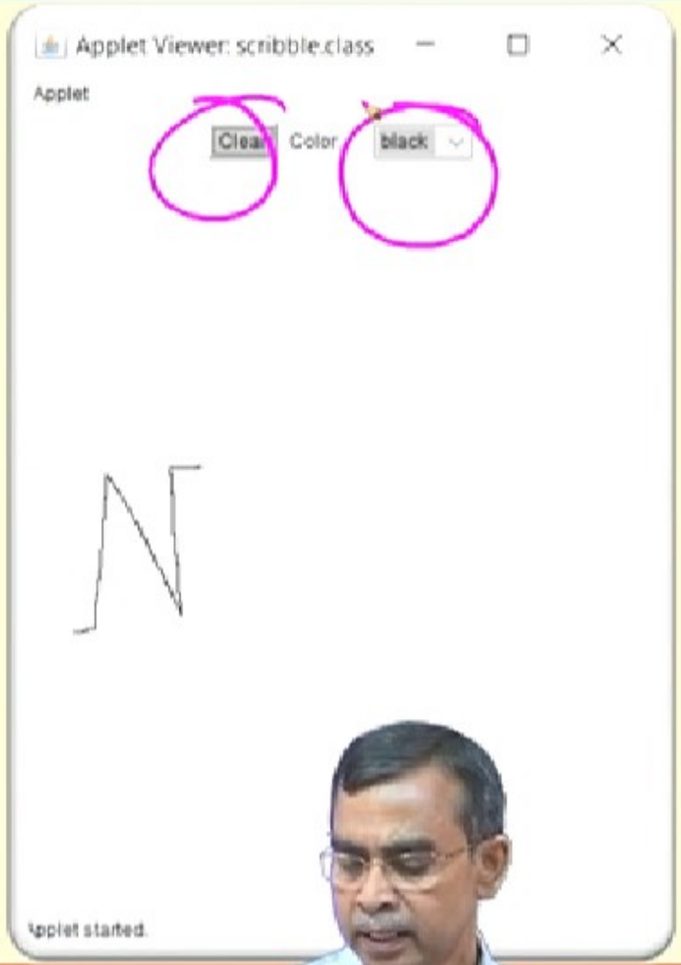
Scribble 2 applet : Another example

```
/*
<applet code="scribble.class" height=500 width=400></applet>
*/
import java.applet.*;
import java.awt.*;

public class Scribble2 extends Applet {
  private int last_x, last_y;                 // Store the last mouse position.
  private Color current_color = Color.black; // Store the current color.
  private Button clear_button;                // The clear button.
  private Choice color_choices;               // The color dropdown list.

  // This method is called to initialize the applet.
  public void init() {
    this.setBackground(Color.white);
    clear_button = new Button("Clear");
    clear_button.setForeground(Color.black);
    clear_button.setBackground(Color.lightGray);
    this.add(clear_button);
```



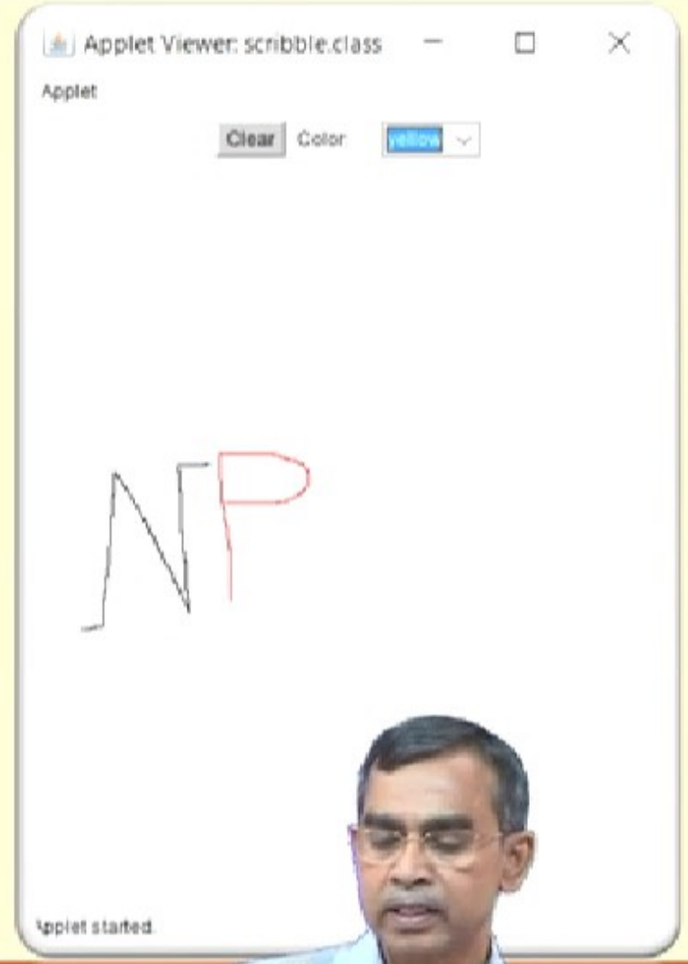
Scribble 2 applet : Another example

```
/*
<applet code="scribble.class" height=500 width=400></applet>
*/
import java.applet.*;
import java.awt.*;

public class Scribble2 extends Applet {
  private int last_x, last_y;                  // Store the last mouse position.
  private Color current_color = Color.black; // Store the current color.
  private Button clear_button;                 // The clear button.
  private Choice color_choices;                // The color dropdown list.

  // This method is called to initialize the applet.
  public void init() {
    this.setBackground(Color.white);
    clear_button = new Button("Clear");
    clear_button.setForeground(Color.black);
    clear_button.setBackground(Color.lightGray);
    this.add(clear_button);
```



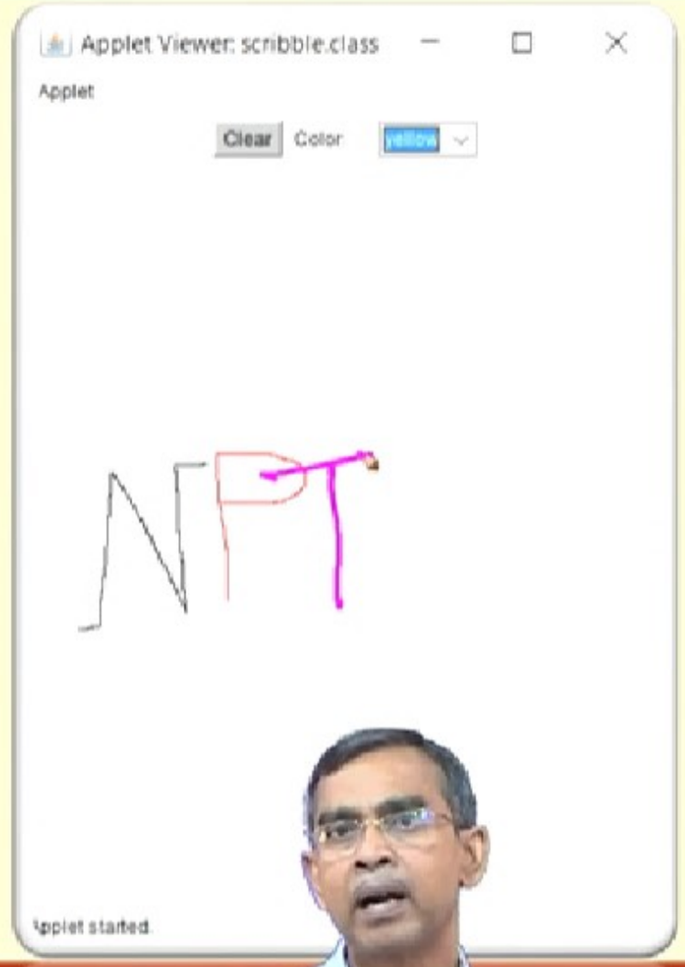
Scribble 2 applet : Another example

```
// Create a menu of colors and add it to the applet.
   // Also set the menus's colors and add a label.
   color_choices = new Choice();
   color_choices.addItem("black");
   color_choices.addItem("red");
   color_choices.addItem("yellow");
   color_choices.addItem("green");
   color_choices.setForeground(Color.black);
   color_choices.setBackground(Color.lightGray);
   this.add(new Label("Color: "));
   this.add(color_choices);
}
// This method is called when the user clicks the mouse to start a scribble.
public boolean mouseDown(Event e, int x, int y){
   last_x = x; last_y = y;
   return true;
}
// This method is called when the user drags the mouse.
public boolean mouseDrag(Event e, int x, int y){
   Graphics g = this.getGraphics();
   g.setColor(current_color);
   g.drawLine(last_x, last_y, x, y);
   last_x = x;
   last_y = y;
   return true;
}
```



Scribble 2 applet : Another example

```
// Create a menu of colors and add it to the applet.
   // Also set the menus's colors and add a label.
   color_choices = new Choice();
   color_choices.addItem("black");
   color_choices.addItem("red");
   color_choices.addItem("yellow");
   color_choices.addItem("green");
   color_choices.setForeground(Color.black);
   color_choices.setBackground(Color.lightGray);
   this.add(new Label("Color: "));
   this.add(color_choices);
}
// This method is called when the user clicks the mouse to start a scribble.
public boolean mouseDown(Event e, int x, int y){
   last_x = x; last_y = y;
   return true;
}
// This method is called when the user drags the mouse.
public boolean mouseDrag(Event e, int x, int y){
   Graphics g = this.getGraphics();
   g.setColor(current_color);
   g.drawLine(last_x, last_y, x, y);
   last_x = x;
   last_y = y;
   return true;
}
```



MouseListener : An example without applet

```
import java.awt.*;

public class MouseListenerExample extends Frame implements MouseListener{
    Label l;
    MouseListenerExample(){
        addMouseListener(this);

        l = new Label();
        l.setBounds(20,50,100,20);
        add(l);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
    public void mouseClicked(MouseEvent e) {l.setText("Mouse Clicked");}
    public void mouseEntered(MouseEvent e) {l.setText("Mouse Entered");}
    public void mouseExited(MouseEvent e)  {l.setText("Mouse Exited"); }
    public void mousePressed(MouseEvent e) {l.setText("Mouse Pressed");}
    public void mouseReleased(MouseEvent e){l.setText("Mouse Released");}

    public static void main(String[] args) {
      new MouseListenerExample();
    }
}
```

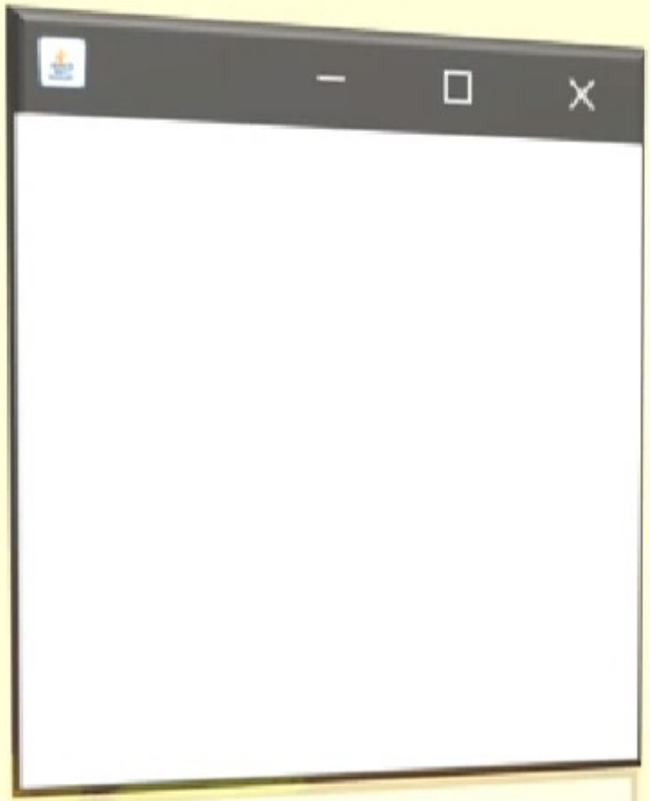
Scribble 2 applet : Another example

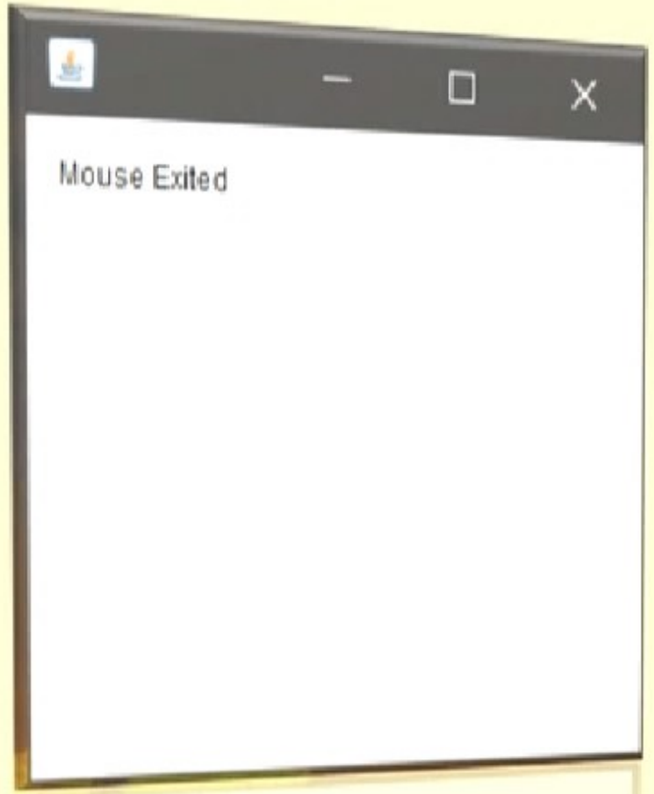
```
// This method is called when the user clicks the button or chooses a color
 public boolean action(Event e, Object arg) {
   // If the Clear button was clicked on, handle it.
   if (e.target == clear_button) {
     Graphics g = this.getGraphics();
     Rectangle r = this.bounds();
     g.setColor(this.getBackground());
     g.fillRect(r.x, r.y, r.width, r.height);
     return true;
   }
   // Otherwise if a color was chosen, handle that
   else if (e.target == color_choices) {
     if (arg.equals("black")) current_color = Color.black;
     else if (arg.equals("red")) current_color = Color.red;
     else if (arg.equals("yellow")) current_color = Color.yellow;
     else if (arg.equals("green")) current_color = Color.green;
     return true;
   }
   // Otherwise, let the superclass handle it.
   else return super.action(event, arg);
 }
}
```

MouseListener : An example without applet

```
import java.awt.*;

public class MouseListenerExample extends Frame implements MouseListener{
    Label l;
    MouseListenerExample(){
        addMouseListener(this);

        l = new Label();
        l.setBounds(20,50,100,20);
        add(l);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
    public void mouseClicked(MouseEvent e) {l.setText("Mouse Clicked");}
    public void mouseEntered(MouseEvent e) {l.setText("Mouse Entered");}
    public void mouseExited(MouseEvent e)  {l.setText("Mouse Exited"); }
    public void mousePressed(MouseEvent e) {l.setText("Mouse Pressed");}
    public void mouseReleased(MouseEvent e){l.setText("Mouse Released");}

    public static void main(String[] args) {
      new MouseListenerExample();
    }
}
```

MouseListener : An example without applet





Handling Keyboard Events

Familiarize with the KeyEvent class

The class which is corresponding of keyboard event is `KeyEvent`. This class is a subset of `InputEvent`. A `KeyEvent` object is generated when keyboard input occurs.

There are three types of key events, which are identified by these integer constants:

Key Events	Description
`KEY_PRESSED`	Whenever any key is pressed, the `KeyPressed()` event hander is called.
`KEY_RELEASED`	Whenever any key is released, the `KeyReleased()` event hander is called.
`KEY_TYPED`	When a character key is typed, `KeyTyped()` event handler is called.

Familiarize with the KeyEvent class

There are many other integer constants that are defined by KeyEvent class:

VK_0 to VK_9 : Defines the ASCII equivalent of the numbers.

VK_A to VK_Z : Defines the ASCII equivalent of letters.

```
VK_ENTER        VK_ESCAPE       VK_CANCEL       VK_UP       VK_DOWN     VK_LEFT
VK_RIGHT        VK_PAGE_DOWN    VK_PAGE_UP      VK_SHIFT    VK_ALT      VK_CONTROL
```

The VK constants specify virtual key codes and are independent of any modifiers, such as control, shift, or alt. The class KeyEvent has two constructors:

```
KeyEvent(Component src, int type, long when, int modifiers, int code)
KeyEvent(Component src, int type, long when, int modifiers, int code, char ch)
```

Familiarize with the KeyEvent class

The class which is corresponding of keyboard event is `KeyEvent`. This class is a subset of `InputEvent`. A `KeyEvent` object is generated when keyboard input occurs.

There are three types of key events, which are identified by these integer constants:

Note:

1. All key press don't result in characters. For example, pressing shift key doesn't generate character.
2. Each time a user presses a key, at least two and often three events are generated.

Familiarize with the KeyEvent class

There are many other integer constants that are defined by KeyEvent class:

VK_0 to VK_9 : Defines the ASCII equivalent of the numbers.

VK_A to VK_Z : Defines the ASCII equivalent of letters.

```
VK_ENTER        VK_ESCAPE       VK_CANCEL       VK_UP       VK_DOWN     VK_LEFT
VK_RIGHT        VK_PAGE_DOWN    VK_PAGE_UP      VK_SHIFT    VK_ALT      VK_CONTROL
```

The VK constants specify virtual key codes and are independent of any modifiers, such as control, shift, or alt. The class KeyEvent has two constructors:

```
KeyEvent(Component src, int type, long when, int modifiers, int code)
KeyEvent(Component src, int type, long when, int modifiers, int code, char ch)
```

Class KeyEvent : methods

Methods	Description
`char getChar()`	Returns character that was entered, for invalid character, it returns CHAR_UNDEFINED
`int getKey()`	Returns the ASCII code of the enterd key.

Implementation : Key event handling

To handle key events, the same steps as in handling mouse events to be followed.

1. Register the following listener in the `init()` method of your applet class.

```
add KeyListner(this);
```

In addition to this, `init()` must include request input focus.

```
requestFocus(); // This method is defined in the component class. If not,
                then the program will not receive any Key events.
```

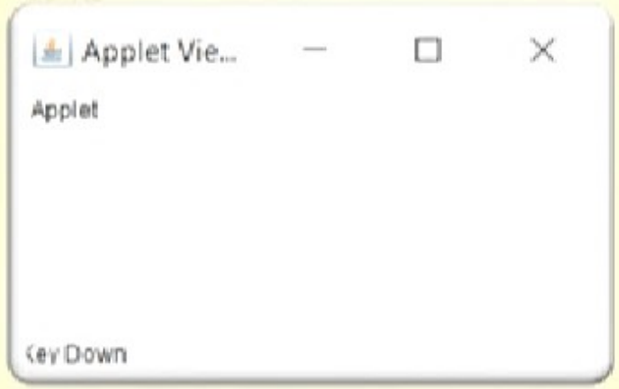
2. Implement the listener methods:

```
KeyPressed(KeyEvent ke)
KeyReleased(KeyEvent ke)
KeyTyped(KeyEvent ke)
```

That's all!

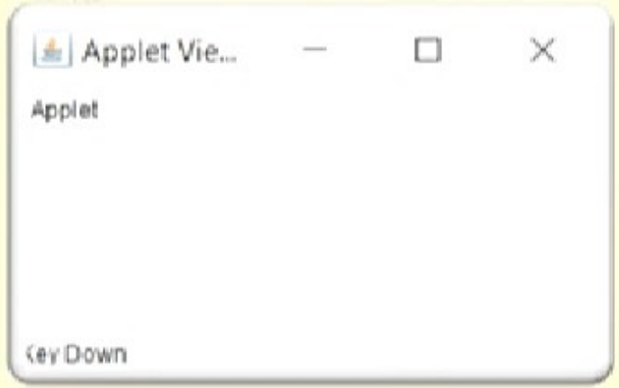
Simple key event : An example

```
import java.awt.*;
import java.awt.event.*;
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);
        requestFocus(); // request input focus
     }
     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();
     }
     public void paint(Graphics g) { // Display keystrokes.
         g.drawString(msg, X, Y);
     }
}
```



Simple key event : An example

```
import java.awt.*;
import java.awt.event.*;
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);
        requestFocus(); // request input focus
    }
    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();
    }
    public void paint(Graphics g) { // Display keystrokes.
        g.drawString(msg, X, Y);
    }
}
```



Virtual key codes : Another example

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class keyevents extends Applet implements KeyListener {
    String msg = ""; int X = 10, Y = 20; // output coordinates
    public void init() {
       addKeyListener(this);  requestFocus();
    }
    public void keyPressed(KeyEvent ke) {
       showStatus("Key Down");
       int key = ke.getKeyCode();
       switch(key) {
         case KeyEvent.VK_F1: msg += "<F1>"; break;
         case KeyEvent.VK_F2: msg += "<F2>"; break;
         case KeyEvent.VK_F3: msg += "<F3>"; break;
         case KeyEvent.VK_PAGE_DOWN: msg += "<PgDn>"; break;
         case KeyEvent.VK_PAGE_UP: msg += "<PgUp>"; break;
         case KeyEvent.VK_LEFT: msg += "<Left Arrow>"; break;
         case KeyEvent.VK_RIGHT: msg += "<Right Arrow>"; break;
       }
       repaint();
    }
    public void keyReleased(KeyEvent ke) { showStatus("Key Up"); }
    public void keyTyped(KeyEvent ke) { msg += ke.getKeyChar(); repaint(); }
    public void paint(Graphics g) { g.drawString(msg, x, y); }
}
```

Virtual key codes : Another example

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class keyevents extends Applet implements KeyListener {
    String msg = ""; int X = 10, Y = 20; // output coordinates
    public void init() {
        addKeyListener(this);  requestFocus();
    }
    public void keyPressed(KeyEvent ke) {
        showStatus("Key Down");
        int key = ke.getKeyCode();
        switch(key) {
          case KeyEvent.VK_F1: msg += "<F1>"; break;
          case KeyEvent.VK_F2: msg += "<F2>"; break;
          case KeyEvent.VK_F3: msg += "<F3>"; break;
          case KeyEvent.VK_PAGE_DOWN: msg += "<PgDn>"; break;
          case KeyEvent.VK_PAGE_UP: msg += "<PgUp>"; break;
          case KeyEvent.VK_LEFT: msg += "<Left Arrow>"; break;
          case KeyEvent.VK_RIGHT: msg += "<Right Arrow>"; break;
        }
        repaint();
    }
    public void keyReleased(KeyEvent ke) { showStatus("Key Up"); }
    public void keyTyped(KeyEvent ke) { msg += ke.getKeyChar(); repaint(); }
    public void paint(Graphics g) { g.drawString(msg, x, y); }
}
```

Thank You