

The background features a dark blue, almost black, sky with faint, thin lines radiating from the top corners. Below the sky is a range of mountains rendered in a wireframe style, with glowing blue lines forming the peaks and ridges. The foreground is a floor made of a grid of glowing purple lines that recede into the distance, creating a sense of perspective.

# Event Handling part 2

# Important Points Revisited

The background of the slide features a digital landscape. The foreground is a floor made of a grid of purple lines that recede into the distance. In the background, there are three mountain-like shapes constructed from a network of blue lines, creating a wireframe effect. The sky is a dark, deep blue with some faint, wispy clouds.



# Important Points Revisited

- To handle a particular kind of event, we have to implement a corresponding interface
- For example
  - Button generates action event, we have to implement **ActionListener** interface to handle action events
  - Window generates window event, we have to implement **WindowListener** interface to handle window events

# Important Points Revisited

- Event Handling Steps
  - Step 1
    - Create components which can generate events
  - Step 2
    - Build component (objects) that can handle events (Event Handlers)
  - Step 3
    - Register handlers with generators



# More Examples

The background of the slide features a digital, futuristic landscape. In the foreground, a perspective grid of glowing purple lines recedes into the distance. In the background, a range of mountains is rendered as a blue wireframe mesh. The sky is a dark, deep blue with faint, wispy clouds and small white specks resembling stars or distant galaxies.

# Handling Mouse Events

The background of the slide features a digital, futuristic landscape. In the foreground, a perspective grid of glowing purple lines recedes into the distance. In the background, a range of mountains is depicted using a blue wireframe or low-poly style. The overall color palette is dark, with deep blues and purples, accented by the bright yellow of the text.



# Handling Mouse Events

- Mouse events can be trapped for any GUI component that inherit from `Component` class. For example, `JPanel`, `JFrame` & `JButton` etc.
- To handle Mouse events, two types of listener interfaces are available
  - `MouseMotionListener`
  - `MouseListener`

# Handling Mouse Events

- MouseMotionListener
  - For processing mouse motion events
  - Mouse motion event is generated when mouse is moved or dragged

```
public interface MouseMotionListener {  
    public void mouseDragged (MouseEvent me);  
    public void mouseMoved (MouseEvent me);  
}
```



# Handling Mouse Events

- MouseListener
  - For processing “interesting” mouse events
  - Mouse event is generated when mouse is
    - Pressed
    - Released
    - clicked (pressed & released without moving the cursor)
    - Enter (mouse cursor enters the bounds of component)
    - Exit (mouse cursor leaves the bounds of component)

# Handling Mouse Events

## Interface MouseListener

```
public interface MouseListener {  
  
    public void mousePressed (MouseEvent me);  
  
    public void mouseClicked (MouseEvent me);  
  
    public void mouseReleased (MouseEvent me);  
  
    public void mouseEntered (MouseEvent me);  
  
    public void mouseExited (MouseEvent me);  
  
}
```



The background features a dark blue, almost black, sky with faint, glowing lines. Below the sky, there are stylized mountains rendered in a wireframe or low-poly style, with lines in shades of teal and blue. The foreground is a grid of lines, also in teal and blue, that recede into the distance, creating a sense of depth. The overall aesthetic is futuristic and digital.

# **Example Code**

## **Handling Mouse Motion Events**

# Steps for Handling Mouse Motion Events

1. ? Create mouse motion events generating components

- `JFrame myFrame = new JFrame();`

2. ? Create Event Handler

- Implement `MouseMotionListener` Interface
- Provide implementaion for the required method

```
public void mouseMoved (MouseEvent e) {  
    // write your code here  
}
```

- provide empty bodies for remaining methods in the interface.



# Steps for Handling Mouse Motion Events

3. Register event generator with event handler

```
myFrame.addMouseMotionListener (this);
```

Object of the Event  
Handler class

# Example Code (cont.)

## Handling Mouse Motion Events

// File EventsEx.java, Code listed in Handout section 12.1

..... // import required packages

public class EventsEx implements MouseMotionListener {

    JFrame f;

    JLabel coord;

    public void initGUI ( ){

        ..... // set layout

        ..... // add label to container

        // registration

**f.addMouseMotionListener(this)**

        .....

    } // end of initGui



# Example Code (cont.)

## Handling Mouse Motion Events

```
// MouseMotionListener event handlers
public void mouseDragged (MouseEvent me) {
    int x = me.getX();
    int y = me.getY();
    coord.setText("Dragged at [" + x + "," + y + "]" );
}
```

```
public void mouseMoved (MouseEvent me) {
    int x = me.getX();
    int y = me.getY();
    coord.setText("Moved at [" + x + "," + y + "]" );
}
```

# Example Code (cont.)

## Handling Mouse Motion Events

```
public static void main (String args[ ]){  
    EventsEx ex = new EventsEx();  
}  
} // end of EventsEx class
```



# Handling Mouse Motion Events

Live Output

The background of the slide features a digital landscape. In the foreground, a perspective grid of lines extends from the bottom towards the horizon, creating a sense of depth. The grid lines are thin and light blue. In the background, a range of mountains is depicted using a wireframe or low-poly style, with lines forming the peaks and valleys. The mountains are rendered in a light blue color. The sky is a solid, dark blue.



# Another Example Handling Window Events



# Handling Window Events

- Want to handle Window Exit event only
  - Why ?
    - When window is closed, control should return back to command prompt
    - But we have already achieved this functionality through following line of code

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```
    - But, what if we want to display some message (Good Bye ) before exiting

# Handling Window Events

## – How ?

- To handle window events, we need to implement “WindowListner” interface.
- “WindowListner” interface contains 7 methods
- We require only one i.e. **windowClosing**
- We have to provide definitions of all methods to make concrete class



# Interface WindowListener

```
public interface WindowListener {  
  
    public void windowActivated (WindowEvent we);  
  
    public void windowClosed (WindowEvent we);  
  
    public void windowClosing (WindowEvent we);  
  
    public void windowDeactivated (WindowEvent we);  
  
    public void windowDeiconified (WindowEvent we);  
  
    public void windowIconified (WindowEvent we);  
  
    public void windowOpened (WindowEvent we);  
  
}
```

# Example Code: Window Exit Handler

## Modification of EventsTest.java

```
// File EventsEx.java, Code listed in Handout section 12.2
.....
public class EventsEx implements MouseMotionListener , WindowListener, {
    JFrame f;
    JLabel coord;

    public void initGUI () {
        ..... // set layouts

        window.addMouseMotionListener(this);

        f.addWindowListener(this);
        .....
    } //end initGUI

    // Event Handlers for MouseMotionListener
    public void mouseDragged(MouseEvent me) {.....}
    public void mouseMoved(MouseEvent) {....}
```



# Example Code: Window Exit Handler

## Modification of EventsTest.java

```
// Event Handlers for WindowListener

public void windowActivated (WindowEvent we) { }

public void windowClosed (WindowEvent we) { }

public void windowClosing (WindowEvent we) {
    JOptionPane.showMessageDialog(null, "Good Bye");
    System.exit(0)
}

public void windowDeactivated (WindowEvent we) { }

public void windowDeiconified (WindowEvent we) { }

public void windowIconified (WindowEvent we) { }

public void windowOpened (WindowEvent we) { }
```

# Example Code: Window Exit Handler

## Modification of EventsTest.java

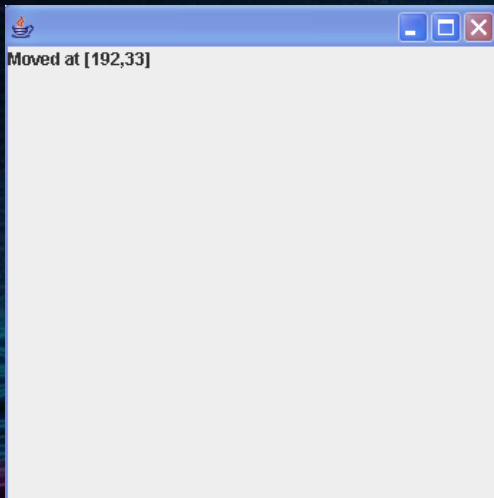
```
public static void main (String args[ ]){  
    EventsEx ex = new EventsEx();  
}  
} // end of EventsEx class
```



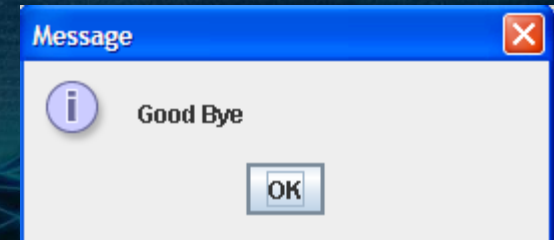
# Example Code: Window Exit Handler

## Modification of EventsTest.java

### Output



When user closes the window,  
Message would be displayed



After pressing Ok button  
Program will exit