GUI & Event Handling

GUI History

When Java 1.0 was introduced, it contained a class library, called the Abstract Window Toolkit (AWT), for basic GUI programming.

The basic AWT library deals with user interface elements by delegating their creation and behavior to the native GUI toolkit on each target platform (Windows, Linux, Macintosh, and so on).

For example, if you used the original AWT to put a text box on a Java window. The resulting program could then, in theory, run on any of these platforms, with the “look-and-feel” of the target platform.

GUI History

In 1996, Netscape created a GUI library they called the IFC (Internet Foundation Classes) that used an entirely different approach. User interface elements, such as buttons, menus, and so on, were painted onto blank windows.

The only functionality required from the underlying windowing system was a way to put up a window and to paint on it. Thus, Netscape's IFC widgets looked and behaved the same no matter which platform the program ran on.

Sun Microsystems worked with Netscape to perfect this approach, creating a user interface library with the code name "Swing." Swing was available as an extension to Java 1.1 and became a part of the standard library in Java 1.2.

Swing is now the official name for the GUI toolkit.

Introduction

JAVA provides a rich set of libraries to create Graphical User Interface.

- AWT (Abstract Window Toolkit).
- SWing

Graphical User Interface (GUI) offers user interaction via some graphical components.

- For example our underlying Operating System also offers GUI via window,frame,Panel, Button, Textfield, TextArea, Listbox, Combobox, Label, Checkbox etc. These all are known as components. Using these components we can create an interactive user interface for an application.

GUI provides result to end user in response to raised events.GUI is entirely based events.

- For example clicking over a button, closing a window, opening a window, typing something in a textarea etc. These activities are known as events.GUI makes it easier for the end user to use an application. It also makes them interesting

Terminologies

Component All the elements like the button, text fields, scroll bars, etc. are called components.

Container The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

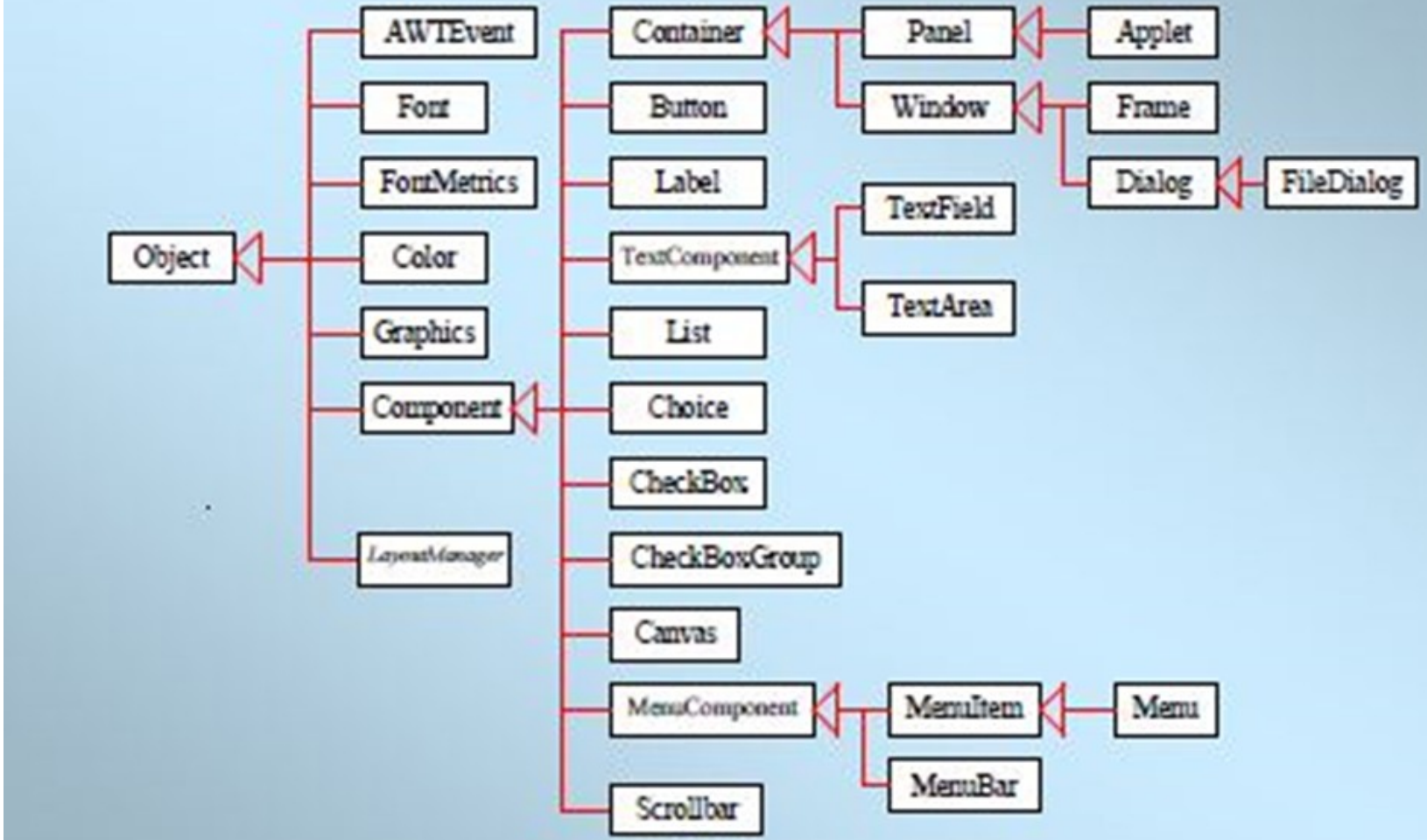
Window The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window. We need to create an instance of Window class to create this container.

Terminologies

Panel Panel provides space in which an application can attach any other components, including other panels.

Frame A Frame is a top-level window with a title and a border. The size of the frame includes any area designated for the border. Frame encapsulates window. It and has a title bar, menu bar, borders, and resizing corners.

Canvas Canvas component represents a blank rectangular area of the screen onto which the application can draw. Application can also trap input events from the use from that blank area of Canvas component.


Object
AWTEvent
Font
FontMetrics
Color
Graphics
Component
LayoutManager
Container
Button
Label
TextComponent
TextField
TextArea
List
Choice
CheckBox
CheckBoxGroup
Canvas
MenuComponent
MenuItem
Menu
MenuBar
Scrollbar
Panel
Applet
Window
Frame
Dialog
FileDialog

Component Class

Useful Methods of Component Class

Method	Description
public void add(Component c)	Inserts a component on this component.
public void setSize(int width,int height)	Sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	Defines the layout manager for the component.
public void setVisible(boolean status)	Changes the visibility of the component, by default false.

Container Class

Component add(Component comp)

Appends the specified component to the end of this container.

float getAlignmentX()

Returns the alignment along the x axis.

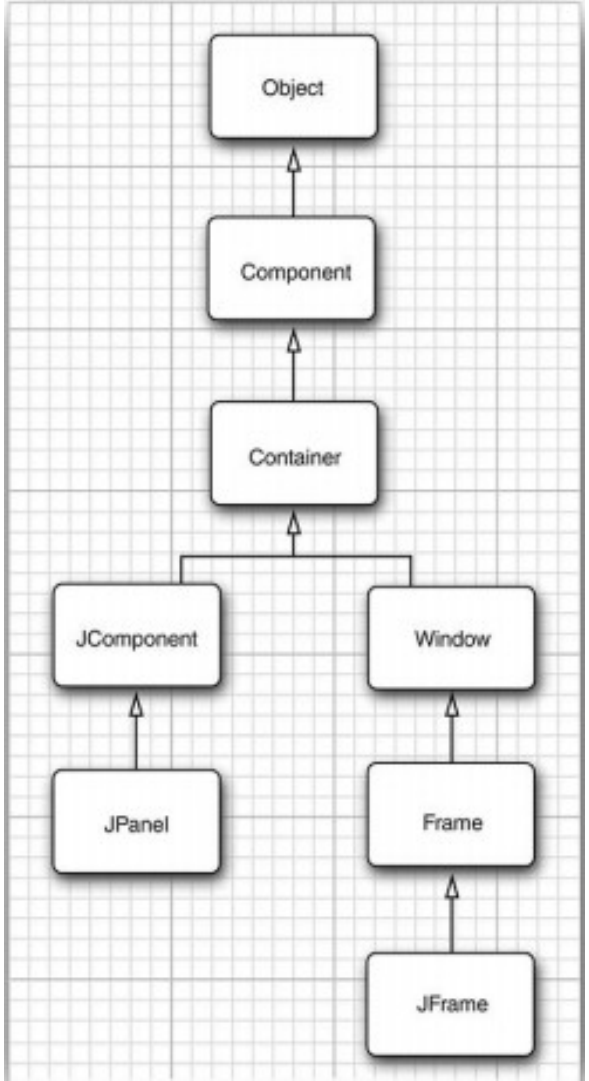
---

float getAlignmentY()

Returns the alignment along the y axis.

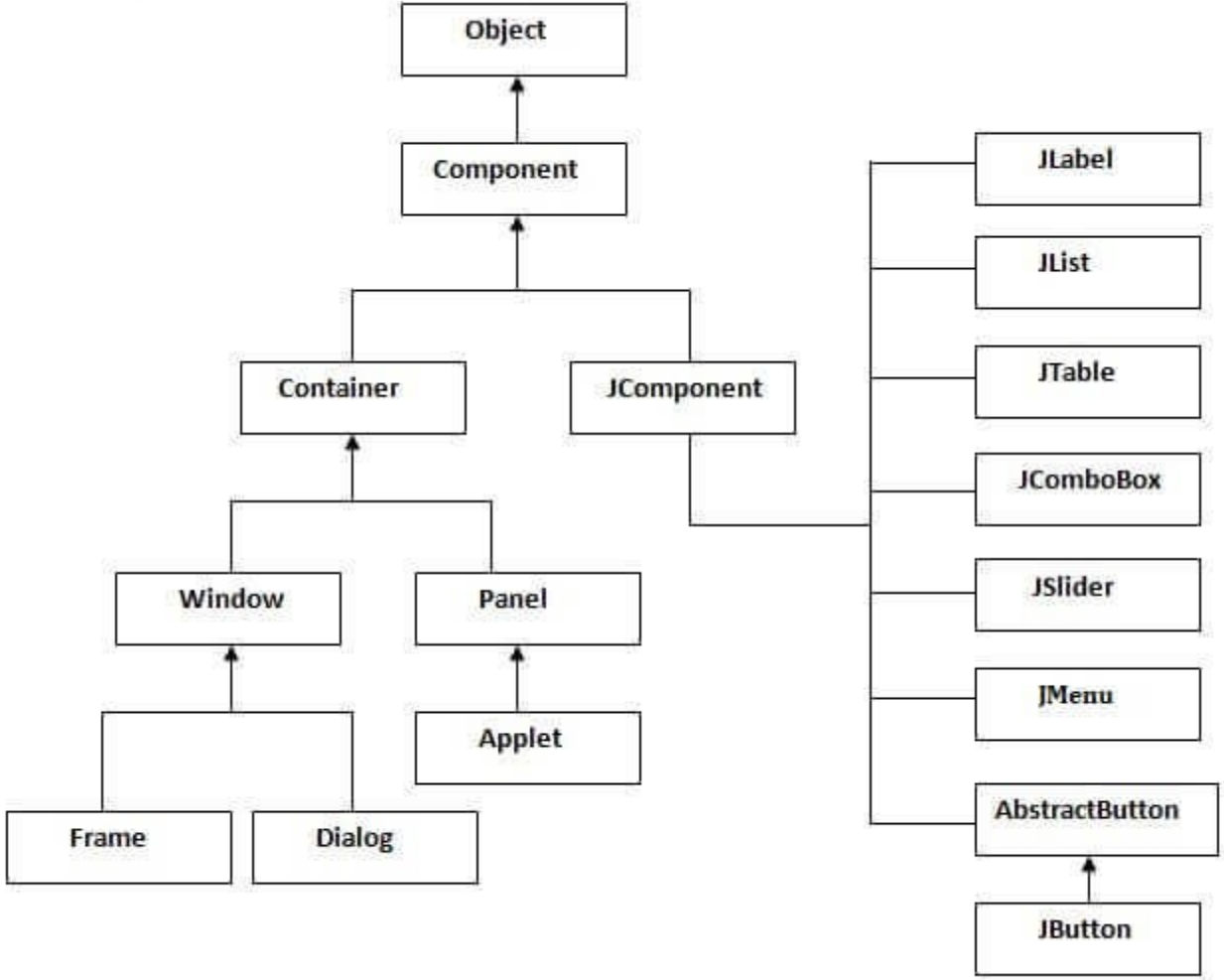
void remove(Component comp)

Removes the specified component from this container.


Object
Component
Container
JComponent
Window
JPanel
Frame
JFrame

Swing

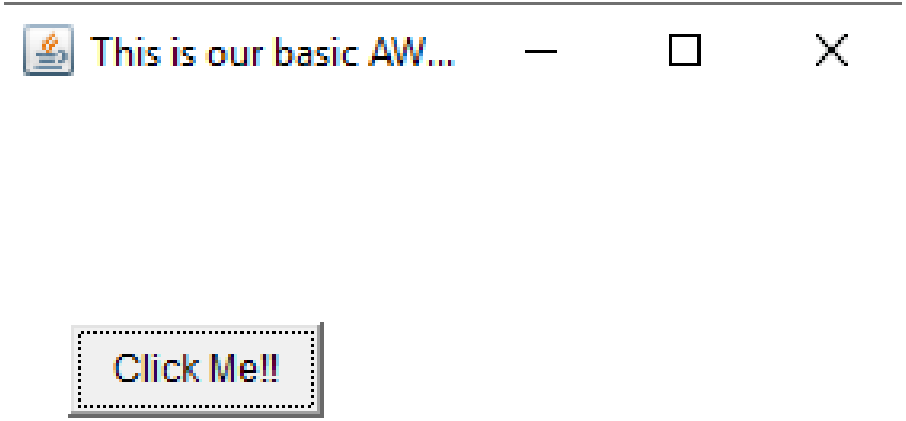
Classes



Create Frame

```
import java.awt.*;
// extending Frame class to our class AWTExample1
public class AWTExample1 extends Frame {

    // initializing using constructor
    AWTExample1() {
        // creating a button
        Button b = new Button("Click Me!!");
        // setting button position on screen
        b.setBounds(30,100,80,30);
        // adding button into frame
        add(b);
        // frame size 300 width and 300 height
        setSize(300,300);
        // setting the title of Frame
        setTitle("This is our basic AWT example");
        // no layout manager
        setLayout(null);
        // now frame will be visible, by default it is not visible
        setVisible(true);
}
public static void main(String args[]) {
// creating instance of Frame class
AWTExample1 f = new AWTExample1();

}
}
```

Create Frame



Swing JFrame


```
import javax.swing.*;
public class FirstSwingExample {
    public static void main(String[] args) {

    JFrame f=new JFrame();//creating instance of JFrame
    JButton b=new JButton("click");//creating instance of JButton
    b.setBounds(130,100,100, 40);//x axis, y axis, width, height
    f.add(b);//adding button in JFrame
    f.setSize(400,500);//400 width and 500 height
    f.setLayout(null);//using no layout managers
    f.setVisible(true);//making the frame visible
}
}
```

Event Handling

What is an Event?

- Change in the state of an object is known as event.
- Events are generated as result of user interaction with the graphical user interface components.
- E.g.
  - clicking on a button
  - moving the mouse
  - entering a character through keyboard
  - selecting an item from list
  - scrolling the page are the activities that causes an event to happen.
- Applets and java graphics Programming are event-driven.
- Events are supported by java.awt.event package.

What is Event Handling?

- Event Handling is the mechanism that controls the event and decides what should happen if an event occurs.
- This mechanism has the code which is known as event handler that is executed when an event occurs.
- For any **event** to occur, the **objects** register themselves as **listeners**.
- No event takes place if there is no listener i.e. nothing happens when an event takes place if there is no listener.
- No matter how many listeners there are, each and every listener is capable of processing an event.
- Java Uses the Delegation Event Model to handle the events.

The Delegation Event Model

The Delegation Event model is defined to handle events in GUI programming languages.

The GUI stands for Graphical User Interface, where a user graphically/visually interacts with the system.

The GUI programming is inherently event-driven; whenever a user initiates an activity such as a mouse activity, clicks, scrolling, etc., each is known as an event that is mapped to a code to respond to functionality to the user. This is known as event handling.

The Delegation Event Model

modern approach for event processing is based on the Delegation Model.

It defines a standard and compatible mechanism to generate and process events.

In this model, a source generates an event and forwards it to one or more listeners.

The listener waits until it receives an event. Once it receives the event, it is processed by the listener and returns it. The UI elements are able to delegate the processing of an event to a separate function.

The key advantage of the Delegation Event Model is that the application logic is completely separated from the interface logic.

The Delegation Event Model

In this model, the listener must be connected with a source to receive the event notifications. Thus, the events will only be received by the listeners who wish to receive them.

Basically, an Event Model is based on the following three components:

- Events
- Events Sources
- Events Listeners

The Delegation Event Model

Events

The Events are the objects that define state change in a source. An event can be generated as a reaction of a user while interacting with GUI elements.

Some of the event generation activities are moving the mouse pointer, clicking on a button, pressing the keyboard key, selecting an item from the list, and so on. We can also consider many other user operations as events.

The Events may also occur that may be not related to user interaction, such as a timer expires, counter exceeded, system failures, or a task is completed, etc. We can define events for any of the applied actions.

The Delegation Event Model

Event Sources

A source is an object that causes and generates an event. It generates an event when the internal state of the object is changed. The sources are allowed to generate several different types of events.

A source must register a listener to receive notifications for a specific event. Each event contains its registration method. Below is an example:

public void addTypeListener (TypeListener e1)

From the above syntax, the Type is the name of the event, and e1 is a reference to the event listener. For example, for a keyboard event listener, the method will be called as **addKeyListener()**. For the mouse event listener, the method will be called as **addMouseMotionListener()**.

The Delegation Event Model

When an event is triggered using the respected source, all the events will be notified to registered listeners and receive the event object. This process is known as event multicasting.

Some listeners allow only one listener to register. Below is an example:

public void addTypeListener(TypeListener e2) throws java.util.TooManyListener sExceptio

From the above syntax, the Type is the name of the event, and e2 is the event listener's reference. When the specified event occurs, it will be notified to the registered listener. This process is known as **unicasting** events.

The Delegation Event Model

A source should contain a method that unregisters a specific type of event from the listener if not needed. Below is an example of the method that will remove the event from the listener.

public void removeTypeListener(TypeListener e2)

From the above syntax, the Type is an event name, and e2 is the reference of the listener. For example, to remove the keyboard listener, the **removeKeyListener()** method will be called.

The Delegation Event Model

Event Listeners

An event listener is an object that is invoked when an event triggers. The listeners require two things;

first, it must be registered with a source; however, it can be registered with several resources to receive notification about the events.

Second, it must implement the methods to receive and process the received notifications.

The methods that deal with the events are defined in a set of interfaces. These interfaces can be found in the java.awt.event package.

For example, the **MouseMotionListener** interface provides two methods when the mouse is dragged and moved.

The Delegation Event Model

Types of Events

The events are categories into the following two categories:

The Foreground Events:

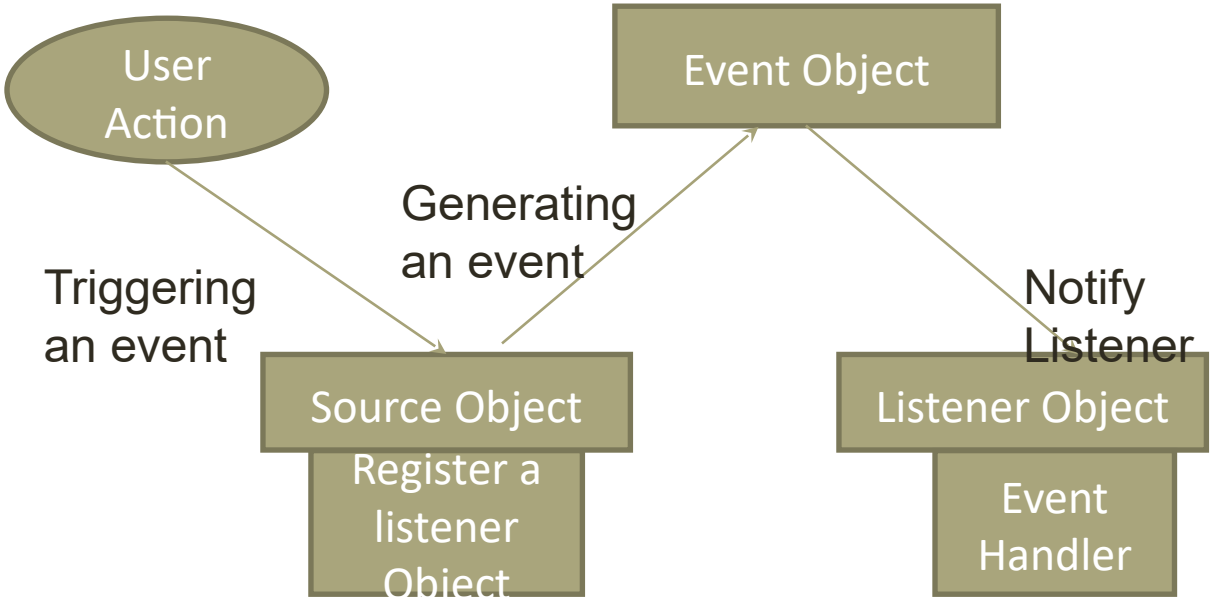
The foreground events are those events that require direct interaction of the user. These types of events are generated as a result of user interaction with the GUI component. For example, clicking on a button, mouse movement, pressing a keyboard key, selecting an option from the list, etc.

The Background Events :

The Background events are those events that result from the interaction of the end-user. For example, an Operating system interrupts system failure (Hardware or Software).

To handle these events, we need an event handling mechanism that provides control over the events and responses.

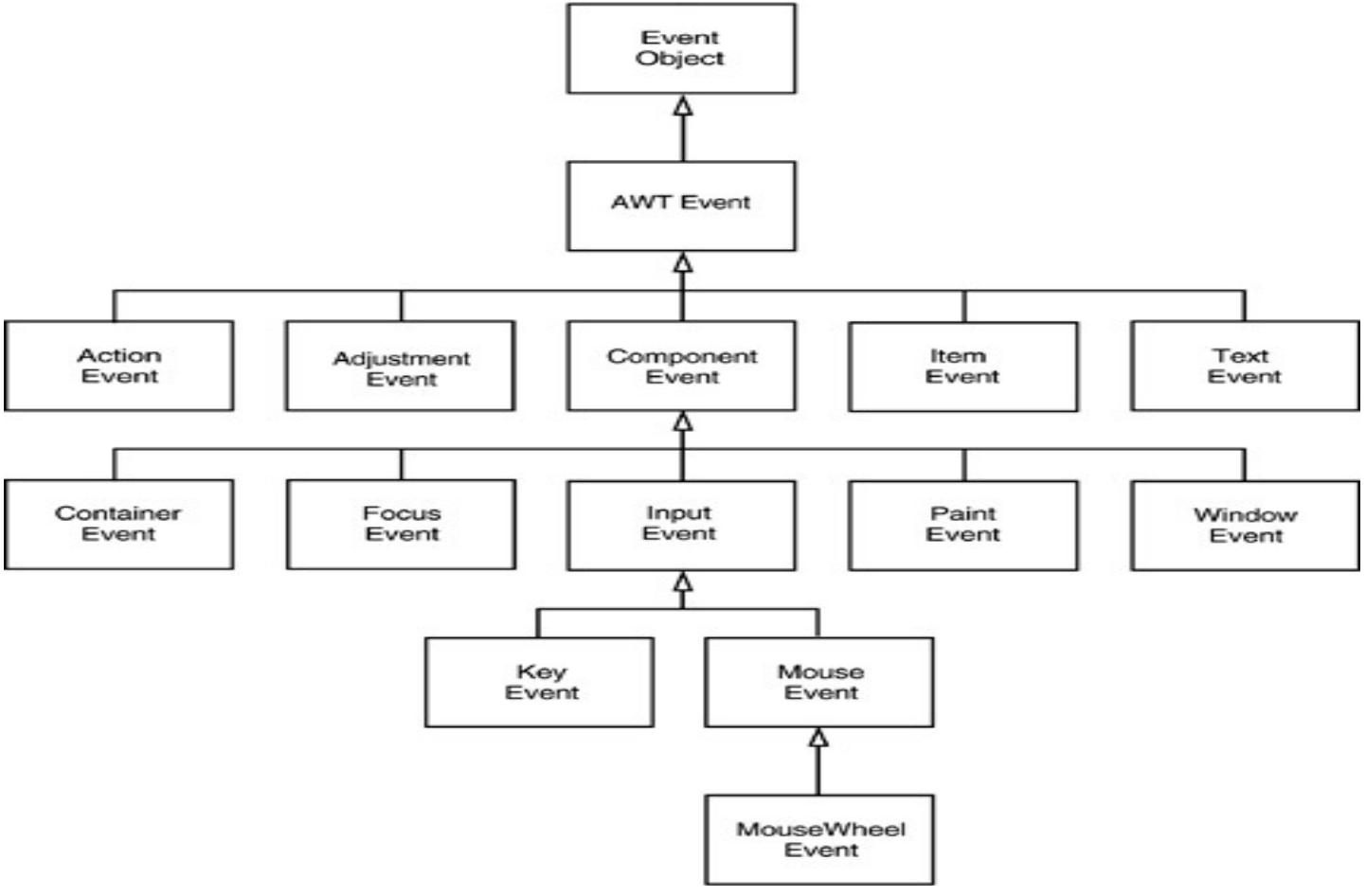
Delegation based Model



Con’t

- In this model ,Listener needs to be registered with the source object so that the listener can receive the event notification.
- This is an efficient way of handling the event because the event notifications are sent only to those listener that want to receive them.

Hierarchy relationship of AWT events



java.util.EventObject class

- Superclass of all events.
- Constructor
  - **EventObject(Object source)**
- Methods
  - **Object getSource()**
  - The object on which the Event initially occurred.
  - **String toString()**
  - Returns a String representation of this EventObject.

java.awt.AWTEvent class

- AWTEvent is a superclass of all AWT events that are handled by the delegation event model.
- This class and its subclasses supercede the original java.awt.Event class.
- **int getID() -** Returns the event type

Event classes in java.awt.Event

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 super class 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.

Con't

Event Class	Description
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. (Added by Java 2, version 1.4)
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.

java.awt.event.ActionEvent class

Constructor

- 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)*

Methods

String getActionCommand( )- Returns the command string associated with this action.

int getModifiers( ) - Returns the modifier keys held down during this action event.

long getWhen( ) - Returns the timestamp of when this event occurred.

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. (Added by Java 2, version 1.4)
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. (Added by Java 2, version 1.4)
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

The ActionListener Interface

The Java ActionListener is notified whenever you click on the button or menu item. It is notified against ActionEvent. The ActionListener interface is found in java.awt.event package. It has only one method: actionPerformed().

actionPerformed() method

The actionPerformed() method is invoked automatically whenever you click on the registered component.

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

The ActionListener Interface

How to write ActionListener

The common approach is to implement the ActionListener. If you implement the ActionListener class, you need to follow 3 steps:

1) Implement the ActionListener interface in the class:

```
public class ActionListenerExample Implements ActionListener
```

2) Register the component with the Listener:

```
component.addActionListener(instanceOfListenerclass);
```

3) Override the actionPerformed() method:

```
public void actionPerformed(ActionEvent e){
        //Write the code here
}
```

The ActionListener Interface

How to write ActionListener

The common approach is to implement the ActionListener. If you implement the ActionListener class, you need to follow 3 steps:

1) Implement the ActionListener interface in the class:

```
public class ActionListenerExample Implements ActionListener
```

2) Register the component with the Listener:

```
component.addActionListener(instanceOfListenerclass);
```

3) Override the actionPerformed() method:

```
public void actionPerformed(ActionEvent e){
        //Write the code here
}
```

Handling Java Swing Button Click Event

Importing Packages

Step 1

In the first step, we need to import all essential packages. In this program, we need to import another new package java.awt.event because we are dealing with event handling and this package provides classes and interfaces that are used for event handling in awt and Swing.

```
//importing all necessary packages
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```

Handling Java Swing Button Click Event

Creating a class MainClass.java

Step 2

In this step, create a class (MainClass.java in this example), and inside that class, there will be our main method.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class MainClass {
    //creating main method
    public static void main(String[] args)
    {

    }
}
```

Handling Java Swing Button Click Event

Creating another class ActionEventDemo.java and create an object of the ActionEventDemo class inside the main method.

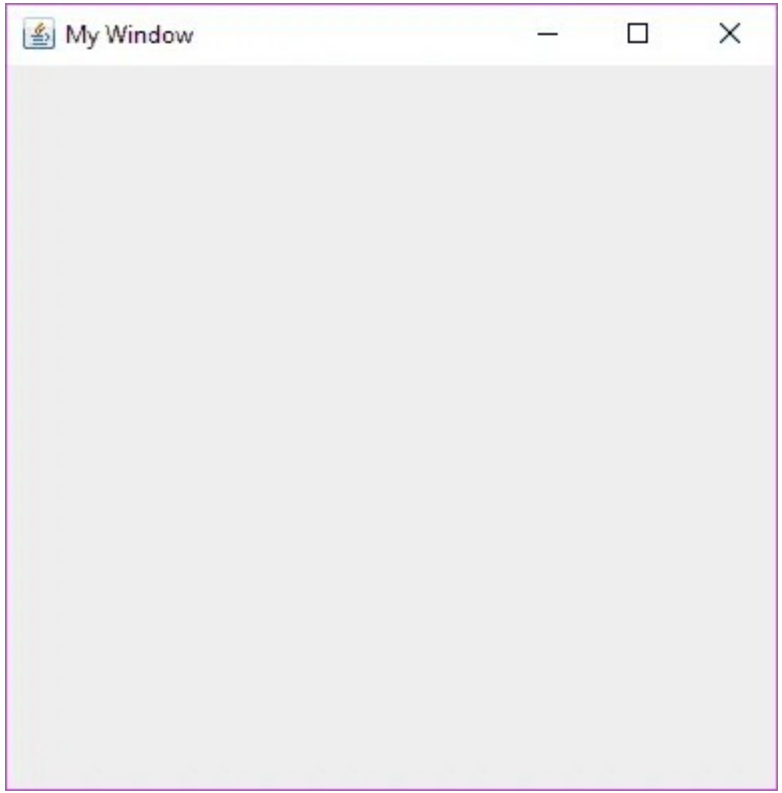
Step 3

1. In this step, create another separate class (ActionEventDemo.java in this example).
2. Now create an object of the JFrame class.
3. Create a user-defined method prepareGUI(), and inside that method we will set the properties of the JFrame class like its title, its location and size, its default close operation, its visibility etc.
4. Now we will create the constructor of the class ActionEventDemo and inside that constructor call the prepareGUI() method.

Handling Java Swing Button Click Event

```
class ActionEventDemo {
    JFrame frame=new JFrame();

    ActionEventDemo(){
        prepareGUI();
    }

    public void prepareGUI(){
        frame.setTitle("My Window");
        frame.getContentPane().setLayout(null);
        frame.setVisible(true);
        frame.setBounds(200,200,400,400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

}

public class MainClass {
    public static void main(String[] args)
    {
        new ActionEventDemo();//Creating object of ActionEventDemo class
    }
}
```

Handling Java Swing Button Click Event



Handling Java Swing Button Click Event

Adding Java Button to JFrame

Step 4

Create an object of the JButton class.
Now again create another user-defined method buttonProperties() and inside that method set the location and size of the JButton using setBounds() method and finally add the JButton to the JFrame using add() method.

Handling Java Swing Button Click Event

Implementing ActionListener Interface

Step 5

Here is an example of how we can implement ActionListener interface into any class.

class classname implements ActionListener

Handling Java Swing Button Click Event

Registering ActionListener to the JButton

Step 6

In this step, we will add or can say register ActionListener to the JButton.
For this, we have to call addActionListner() method using the object of the JButton class.

The parameter of the addActionListener() method is the object of that class in which ActionListener interface is implemented or can say in which we have defined actionPerformed() method. So if we are in the same class in which ActionListener interface is implemented, then we will pass this as an argument.

Handling Java Swing Button Click Event

Performing Action Event

Step 7

Now we want that if we click on the button the background color of the frame's content pane should be changed. For this, we will write the desired codes inside the actionPerformed() method. Which means the behaviour we want in response to the action is coded inside the actionPerformed() method.

Handling Java Swing Button Click Event

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class ActionEventDemo implements ActionListener {
    JFrame frame=new JFrame();
    JButton button=new JButton("Click Me");

    ActionEventDemo(){
        prepareGUI();
        buttonProperties();
    }
```

Handling Java Swing Button Click Event

```
    public void prepareGUI(){
        frame.setTitle("My Window");
        frame.getContentPane().setLayout(null);
        frame.setVisible(true);
        frame.setBounds(200,200,400,400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public void buttonProperties(){
        button.setBounds(130,200,100,40);
        frame.add(button);
        button.addActionListener(this);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        //Changing Background Color
        frame.getContentPane().setBackground(Color.pink);

    }
}

public class MainClass2 {
    public static void main(String[] args)
    {
        new ActionEventDemo();
    }
}
```

Handling Java Swing Button Click Event