Java - Applets

Java programs can be divided into two categories :-

i) Applications and
ii) Applets.

Applications are the programs that contain main( ) method and applets are the programs that do not contain main( ) method. Applications can be executed with a Java interpreter from the command line(with java command). Applets need a browser to execute.

Applet: Applet is a Java program that runs on a browser.

Basic differences between an Application and Applet:

S.No.	Property	Application	Applet
1.	main( ) method	exists	does not exist
2.	Nature	stand-alone programs	can't be stand-alone
3.	Execution	needs a Java interpreter	needs a browser like Netscape, chrome etc.
4.	Security	does not need any security needs top-most security for	hard disk files
5.	Restrictions	no hard disk accessing restrictions	can't access hard disk files, by default
6.	Extra Software	can share any software available	can't share eg. ActiveX controls in the system
7.	Plug-ins	latest additions of software can be embedded through plug-ins	can't use plug-ins directly including browser plug-ins that are incorporated on the user's system.

Java - Applet Life Cycle Methods

- Just like threads, applets too got a life cycle (different stages of execution) between their birth and death.

Life Cycle of an Applet:
Applet goes through different stages of execution between its birth and death. Many of these stages are called implicitly by the browser runtime environment.

init( ) : This is the initialization method to set the properties of an applet. All the initialization statements(like we do in a constructor) are included here. This is the first method called when the applet is loaded. In this stage, applet keeps links with system resources like memory etc.

start( ) : This starts the applet running(like start( ) of a thread). This method is also used to restart the applet after it has stopped. This method is also called when applet is deiconified.

paint( ) : This method is not required directly for an applet's execution. The paint( ) method Graphics class, paints the applet when ever it is started or restarted or resized. Always start( ) method implicitly calls paint( ) method.

stop( ) : Stopping an applet interrupts its execution but it leaves its resources intact so that the applet can be started again easily. We can stop an applet explicitly when a pause is required. Iconification of an applet calls stop( ) method.

destroy( ) : When we close the applet, it is destroyed. This method frees the applet with its resources(returned to the system) --- memory, processor-time, swap disk space etc.

If we do not define any of the above methods in our program, the Applet class supplies its default methods(does not contain any functionality) to the browser for the normal applet execution.

Java – Applet viewer

- Appletviewer is a minimal browser that comes with jdk with which we can run an applet.

  When we do not have a browser to test or run an applet, jdk software provides an applet viewer to run and view the applets. Applet viewer is a small software and understands very minimum HTML tags.

  It is not a full-fledged software and we cannot connect to Internet with it. If Demo.html is our html file, the following statement, runs the program :

  c: \ windows \ applet > appletviewer Demo.html

  How to run an applet and write the HTML tags required to run an applet in a browser ?

  Applet is a Java program without a main( ) method. We need a browser to run an applet.

  Steps to run an applet :
  1. Writing the source code :
  Let us assume the name of the applet as Demo.java. Open an editor and write the source code. The user class should extend Applet class(available in java.applet package). Save the file.

  2. Compilation :
  To compile a Java program, we do not require a main( ) method. So, compile the above file Demo.java and obtain .class file - Demo.class.

  3. Create a HTML file :
  To run the above Demo.class file, we need a HTML file, because .class file cannot be opened directly in a browser. Let the name of the HTML file be Hello.html. Embed the .class file of the applet in HTML file by using APPLET tag.

  APPLET tag is useful to place an applet inside a Web page and describe its attributes and environment. The tag tells he browser where to find the applet to download.

  Java provides a set of methods(like getParameter( )) that an applet uses to obtain information about its environment - including embedded tag attributes and their values. We can use these attributes to configure the applet.

  The following is the APPLET tag :

  < APPLET CODE = "Demo.class" WIDTH = 300 HEIGHT = 200 > < / APPLET >

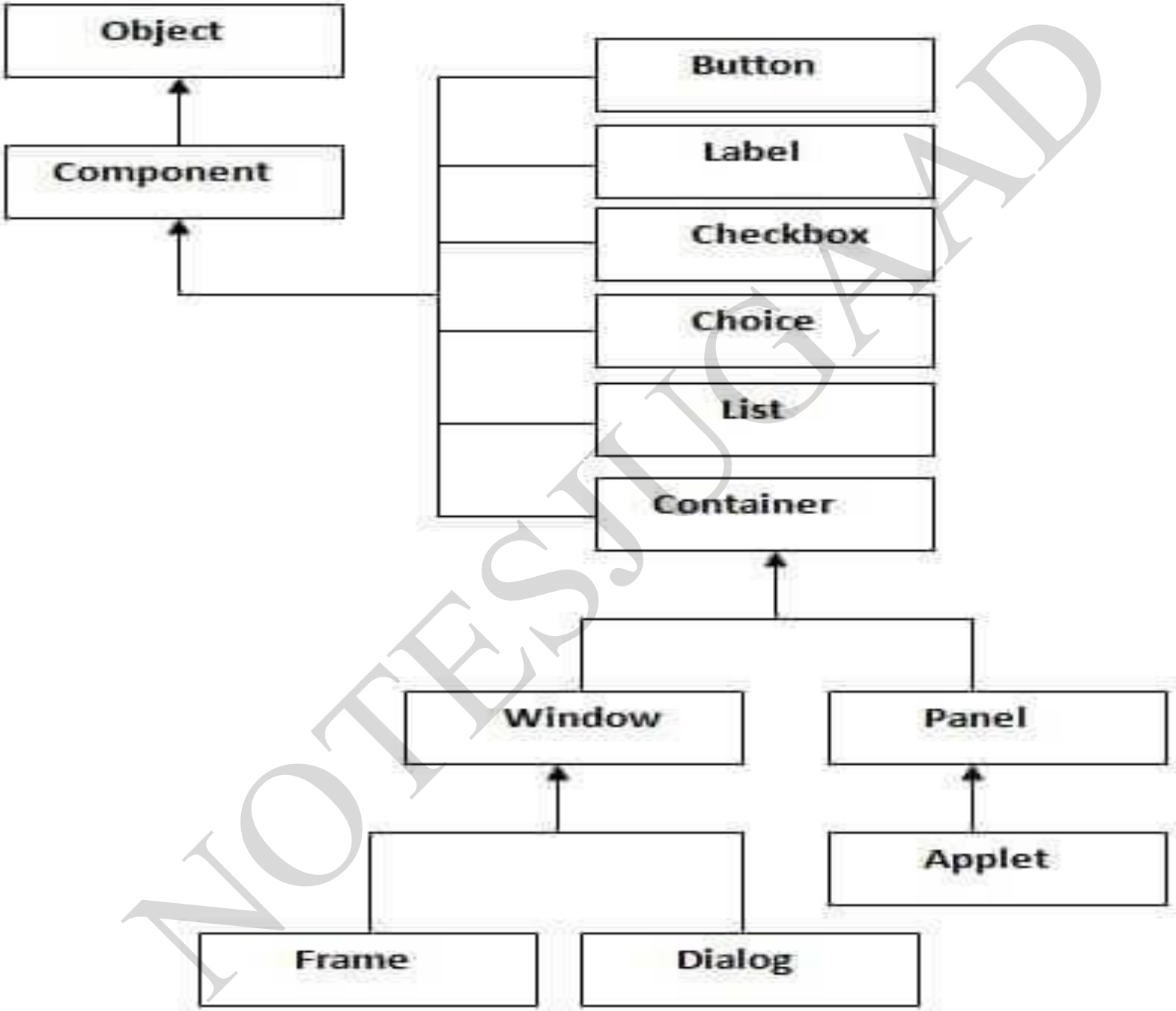
JAVA AWT

- **Java AWT** (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.
- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT Hierarchy

- The hierarchy of Java AWT classes are given below.



- 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.

Panel

- The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame

- The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Java AWT Example

- To create simple awt example, you need a frame. There are two ways to create a frame in AWT.
- **By extending Frame class (inheritance)**
- **By creating the object of Frame class (association)**

AWT Example by Inheritance

```
import java.awt.*;
class First extends Frame{
First(){
Button b=new Button("click me");
b.setBounds(30,100,80,30);// setting button position
add(b);//adding button into frame
setSize(300,300);//frame size 300 width and 300 height
setLayout(null);//no layout manager
setVisible(true);//now frame will be visible, by default not visible
}
public static void main(String args[]){
First f=new First();
}}
```

AWT Example by Association

```
import java.awt.*;
class First2{
First2(){
Frame f=new Frame();
Button b=new Button("click me");
b.setBounds(30,50,80,30);
f.add(b);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[]){
First2 f=new First2();
}}
```

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.

Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener

Steps to perform Event Handling

Following steps are required to perform event handling:

- Register the component with the Listener
- Registration Methods
- For registering the component with the Listener, many classes provide the registration methods. For example:
- **Button**
  - public void addActionListener(ActionListener a){}
- **MenuItem**
  - public void addActionListener(ActionListener a){}
- **TextField**
  - public void addActionListener(ActionListener a){}
  - public void addTextListener(TextListener a){}

- **TextArea**
  - public void addTextListener(TextListener a){}
- **Checkbox**
  - public void addItemListener(ItemListener a){}
- **Choice**
  - public void addItemListener(ItemListener a){}
- **List**
  - public void addActionListener(ActionListener a){}
  - public void addItemListener(ItemListener a){}

Java event handling by implementing ActionListener

```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame implements ActionListener{
TextField tf;
AEvent(){

//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);
```

Java event handling by implementing ActionListener

- //register listener
- b.addActionListener(**this**);//passing current instance
-
- //add components and set size, layout and visibility
- add(b);add(tf);
- setSize(300,300);
- setLayout(**null**);
- setVisible(**true**);
- }

Java event handling by implementing ActionListener

- **public void** actionPerformed(ActionEvent e){
- tf.setText("Welcome");
- }
- **public static void** main(String args[]){
- **new** AEvent();
- }
- }

GUI Event Handling

What is an Event?

- GUI components communicate with the rest of the applications through events.
- The source of an event is the component that causes that event to occur.
- The listener of an event is an object that receives the event and processes it appropriately.

Handling Events

- Every time the user types a character or clicks the mouse, an event occurs.
- Any object can be notified of any particular event.
- To be notified for an event,
  - The object has to be registered as an event listener on the appropriate event source.
  - The object has to implement the appropriate interface.

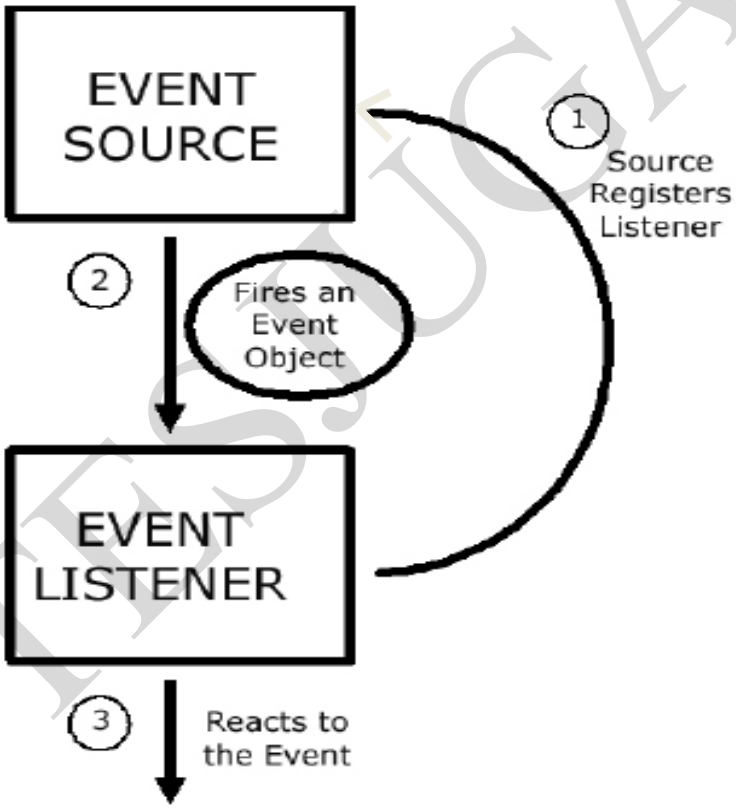
An example of Event Handling

```
public class SwingApplication implements
   ActionListener {
   ...
   JButton button = new JButton("I'm a Swing
   button!");
   button.addActionListener(this);
   ....
   public void actionPerformed(ActionEvent e) {
   numClicks++;
   label.setText(labelPrefix + numClicks);
   }
}
```

The Event Handling process

- When an event is triggered, the JAVA runtime first determines its source and type.
- If a listener for this type of event is registered with the source, an event object is created.
- For each listener to this type of an event, the JAVA runtime invokes the appropriate event handling method to the listener and passes the event object as the parameter.

The Event Handling Process (contd..)



What does an Event Handler require?

- It just looks for 3 pieces of code!
- First, in the declaration of the event handler class, one line of code must specify that the class implements either a listener interface or extends a class that implements a listener interface.

```
public class DemoClass implements ActionListener {
```

What does an Event Handler require? (contd..)

- Second, it looks for a line of code which registers an instance of the event handler class as a listener of one or more components because, as mentioned earlier, the object must be registered as an event listener.

```
anyComponent.addActionListener(instanceOf
DemoClass);
```

What does an Event Handler require? (contd..)

- Third, the event handler must have a piece of code that implements the methods in the listener interface.

```
public void actionPerformed(ActionEvent e) {
  ...//code that reacts to the action...
 }
```

Types of Events

- Below, are some of the many kinds of events, swing components generate.

Act causing Event	Listener Type
User clicks a button, presses Enter, typing in text field	ActionListener
User closes a frame	WindowListener
Clicking a mouse button, while the cursor is over a component	MouseListener

Types of Events (contd..)

Act causing Event	Listener Type
User moving the mouse over a component	MouseMotionListener
Component becomes visible	ComponentListener
Table or list selection changes	ListSelectionListener

The Event classes

- An event object has an event class as its reference data type.
- The Event object class
  - Defined in the java.util package.
- The AWT Event class
  - An immediate subclass of EventObject.
  - Defined in java.awt package.
  - Root of all AWT based events.

Event Listeners

- Event listeners are the classes that implement the

  <type>Listener interfaces.

  Example:

  1. ActionListener receives action events
  2. MouseListener receives mouse events.

The following slides give you a brief overview on some of the listener types.

The ActionListener Method

- It contains exactly one method.

  Example:

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

  The above code contains the handler for the ActionEvent e that occurred.

The MouseListener Methods

- Event handling when the mouse is clicked.

```
public void mouseClicked(MouseEvent e)
```

- Event handling when the mouse enters a component.

```
public void mouseEntered(MouseEvent e)
```

- Event handling when the mouse exits a component.

```
public void mouseExited(MouseEvent e)
```

The MouseListener Methods (contd..)

- Event handling when the mouse button is pressed on a component.

```
public void mousePressed(MouseEvent e)
```

- Event handling when the mouse button is released on a component.

```
public void mouseReleased(MouseEvent e)
```

The MouseMotionListener Methods

- Invoked when the mouse button is pressed over a component and dragged. Called several times as the mouse is dragged

```
public void mouseDragged(MouseEvent e)
```

- Invoked when the mouse cursor has been moved onto a component but no buttons have been pushed.

```
public void mouseMoved(MouseEvent e)
```

The WindowListener Methods

- Invoked when the window object is opened.

```
public void windowOpened(WindowEvent e)
```

- Invoked when the user attempts to close the window object from the object's system menu.

```
public void windowClosing(WindowEvent e)
```

The WindowListener Methods (contd..)

- Invoked when the window object is closed as a result of calling dispose (release of resources used by the source).

```
public void windowClosed(WindowEvent e)
```

- Invoked when the window is set to be the active window.

```
public void windowActivated(WindowEvent e)
```

The WindowListener Methods (contd..)

- Invoked when the window object is no longer the active window

```
public void windowDeactivated(WindowEvent e)
```

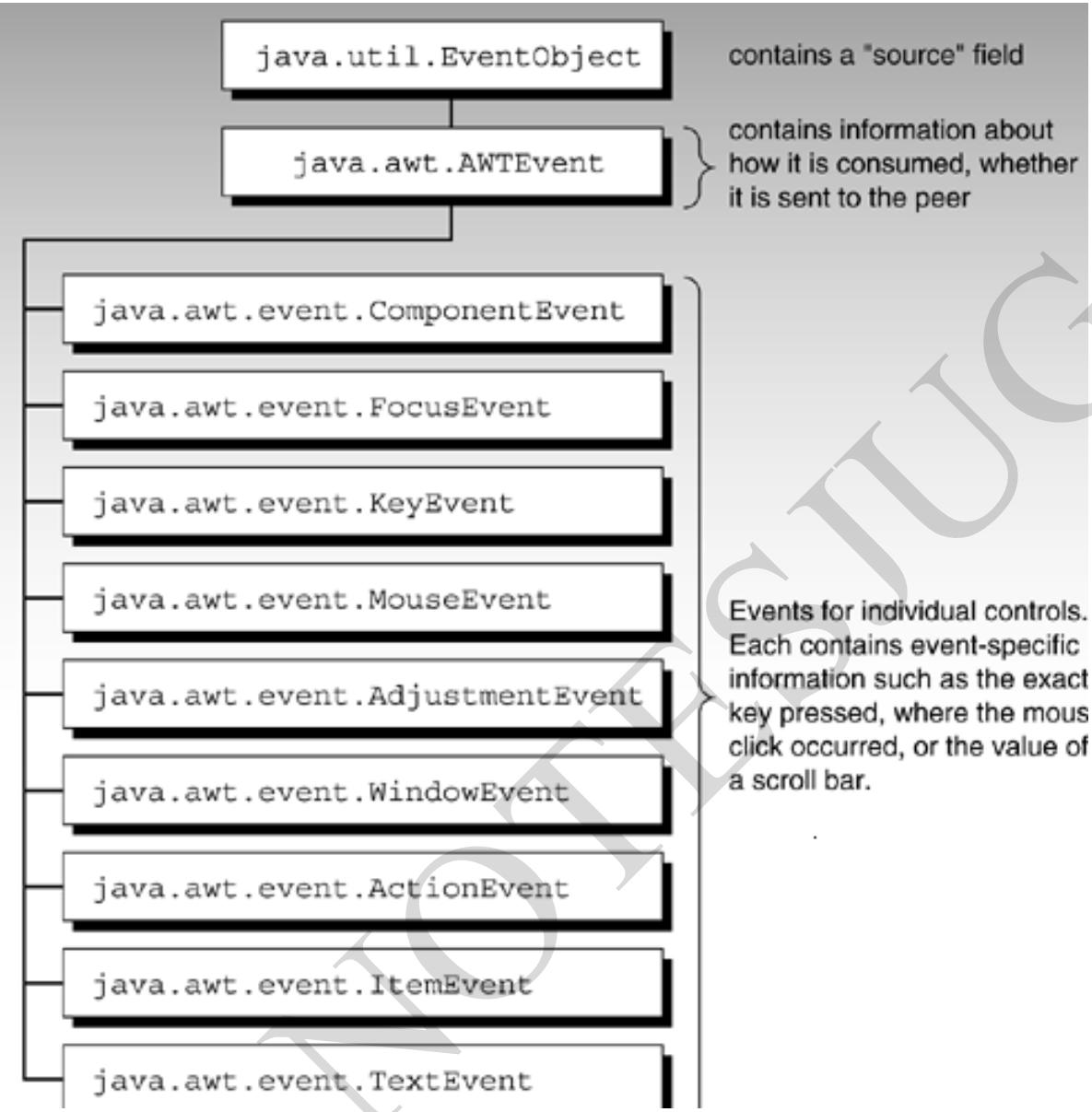
- Invoked when the window is minimized.

```
public void windowIconified(WindowEvent e)
```

- Invoked when the window is changed from the minimized state to the normal state.

```
public void windowDeconified(WindowEvent e)
```

Hierarchy of event **objects**



Note: The number of event objects is much greater then specified in diagram…Due to space constraints, only some of them are represented in the figure

Courtesy: Safari.oreilly.com

Additional Listener Types

- Change Listener
- Container Listener
- Document Listener
- Focus Listener
- Internal Frame Listener
- Item Listener
- Key Listener
- Property Change Listener
- Table Model Listener

The main purpose of the last few slides is to give you an idea as to how you can use event handlers in your programs. It is beyond the scope of the O'Reilly book to cover every event handler. See the JAVA tutorials for more information.

Adapter classes for Event Handling.

- Why do you need adapter classes?
  - Implementing all the methods of an interface involves a lot of work.
  - If you are interested in only using some methods of the interface.
- Adapter classes
  - Built-in in JAVA
  - Implement all the methods of each listener interface with more than one method.
  - Implementation of all empty methods

Adapter classes - an Illustration.

- Consider, you create a class that implements a MouseListener interface, where you require only a couple of methods to be implemented. If your class directly implements the MouseListener, you *must* implement all five methods of this interface.
- Methods for those events you don't care about can have empty bodies

Illustration (contd..)

```
public class MyClass implements MouseListener {
   ... someObject.addMouseListener(this);
   /* Empty method definition. */
   public void mousePressed(MouseEvent e) { }
   /* Empty method definition. */
   public void mouseReleased(MouseEvent e) { }
   /* Empty method definition. */
   public void mouseEntered(MouseEvent e) { }
   /* Empty method definition. */
   public void mouseExited(MouseEvent e) { }
   public void mouseClicked(MouseEvent e) {
   //Event listener implementation goes here...
   }
}
```

Illustration (contd..)

- What is the result?
  - The resulting collection of empty bodies can make the code harder to read and maintain.
- To help you avoid implementing empty bodies, the API generally includes an *adapter* class for each listener interface with more than one method. For example, the MouseAdapter class implements the MouseListener interface.

How to use an Adapter class?

```
/* Using an adapter class
 */
public class MyClass extends MouseAdapter {
....
   someObject.addMouseListener(this);
....
   public void mouseClicked(MouseEvent e) {
     ...//Event listener implementation goes
        // here
   }
}
```

Using Inner classes for Event Handling

- Consider that you want to use an adapter class but you don't want your public class to inherit from the adapter class.
- For example, you write an applet with some code to handle mouse events. As you know, JAVA does not permit multiple inheritance and hence your class cannot extend both the Applet and MouseAdapter classes.

Using Inner classes (contd..)

- Use a class inside your Applet subclass that extends the MouseAdapter class.

```
public class MyClass extends Applet { ...
 someObject.addMouseListener(new
                              MyAdapter());
 ...
  class MyAdapter extends MouseAdapter {   public
 void mouseClicked(MouseEvent e) {
//Event listener implementation here... }
 }
}
```

Creating GUI applications with Event Handling.

- Guidelines:
  1. Create a GUI class
     - Describes the appearance of your GUI application.
  2. Create a class implementing the appropriate listener interface
     - May refer to the same class as step 1.

Creating GUI applications with Event Handling (contd..)

3. In the implementing class
   - Override all methods of the appropriate listener interface.
   - Describe in each method how you want to handle the events.
   - May give empty implementations for the methods you don't need.

Creating GUI applications with Event Handling (contd..)

4. Register the listener object with the source
   - The object is an instantiation of the listener class specified in step 2.
   - Use the *add<Type>Listener* method.

Design Considerations

- The most important rule to keep in mind about event listeners is that they must execute quickly. Because, all drawing and event-listening methods are executed in the same thread, a slow event listener might make the program seem unresponsive. So, consider the performance issues also when you create event handlers in your programs.

Design Considerations

- You can have choices on how the event listener has to be implemented. Because, one particular solution might not fit in all situations.

  For example, you might choose to implement separate classes for different types of listeners. This might be a relatively easy architecture to maintain, but many classes can also result in reduced performance .

Common Event-Handling Issues

1. You are trying to handle certain events from a component, but it doesn't generate the events it should.
   - Make sure you have registered the right kind of listener to detect the events.
   - Make sure you have registered the listener on the right object.
   - Make sure you have implemented the event handler correctly, especially, the method signatures.

Common Event-Handling Issues (contd..)

2. Your combo box isn't generating low level events like focus events.
   - Since combo boxes are compound components, i.e., components implemented using multiple components, combo-boxes do not fire the low-level events that simple components fire.

Common Event-Handling Issues (contd..)

3. The document for an editor pane is not triggering document events.
   - The document instance for an editor pane might change when loading text from a URL. Thus your listeners might be listening for events on an unused document.
   - Hence, make sure that the code adjusts for possible changes to the document if your program dynamically loads text into an editor pane.

References

- *Jia, Xiaoping, Object Oriented Software Development Using Java*. Addison Wesley, 2003
- http://java.sun.com/docs/books/tutorial/uiswing/events/index.html
- http://java.sun.com/docs/books/tutorial/uiswing/learn/example2.html#handlingEvents
- http://safari.oreilly.com/0672315467/ch09

Java Swing

Java Swing

- Java Swing is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.
- Unlike AWT, Java Swing provides platform-independent and lightweight components.

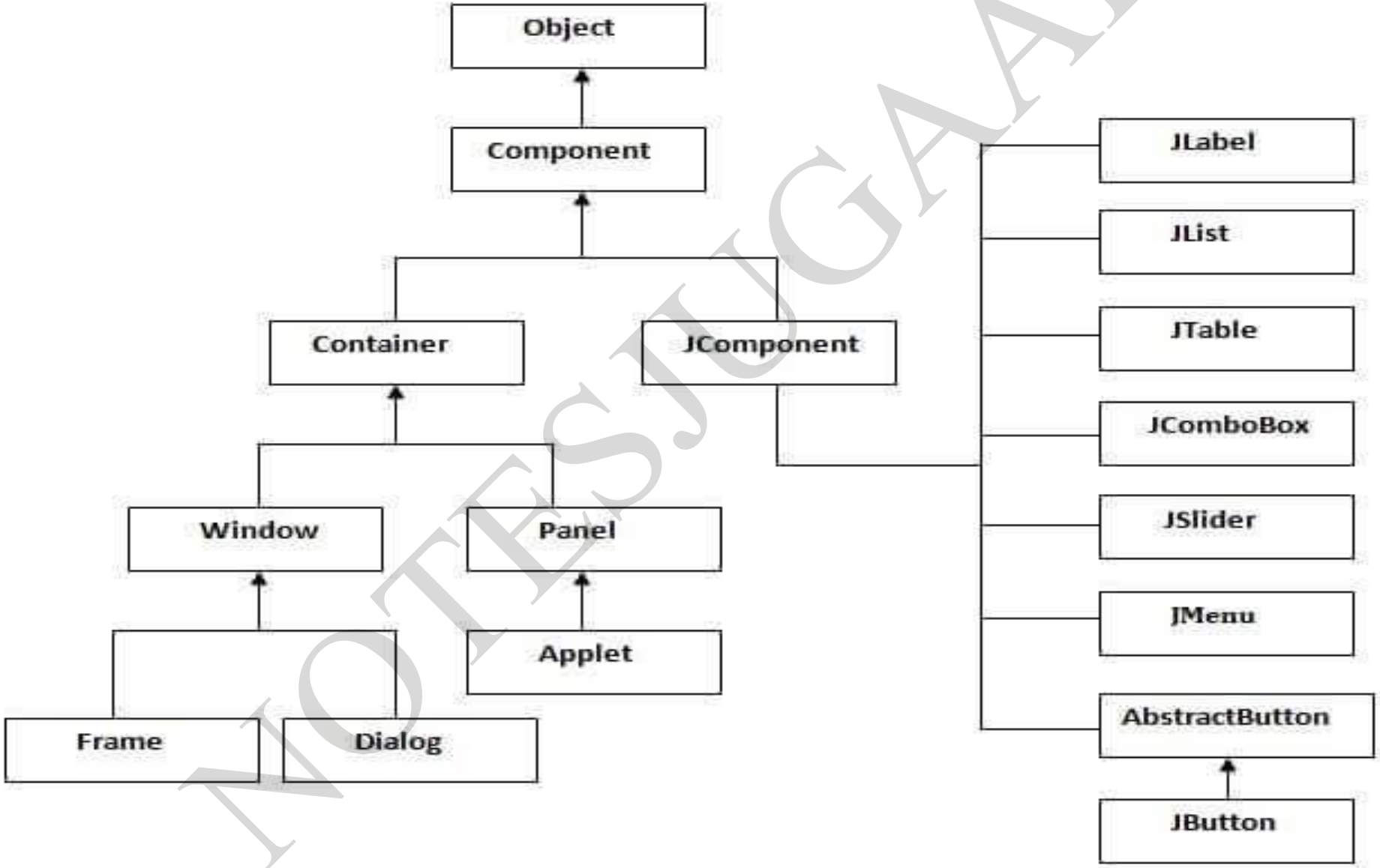
Java Swing

The **javax.swing** package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc

Difference between AWT and Swing

No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC**.

Hierarchy of Java Swing classes



Simple Java Swing Example

- **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 J Button
- b.setBounds(130,100,100, 40);//x axis, y axis, width, he ight
-

```
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
}
}
```

Example of Swing by Association inside constructor

- **import** javax.swing.*;
- **public class** Simple {
- JFrame f;
- Simple(){
- f=**new** JFrame();//creating instance of JFrame
-
- JButton b=**new** JButton("click");//creating instance of JButton
- b.setBounds(130,100,100, 40);

```
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
}

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

Simple example of Swing by inheritance

- **import** javax.swing.*;
- **public class** Simple2 **extends** JFrame{//inheriting JFrame
- JFrame f;
- Simple2(){
- JButton b=**new** JButton("click");//create button
- b.setBounds(130,100,100, 40);
-
- add(b);//adding button on frame

- setSize(400,500);
- setLayout(**null**);
- setVisible(**true**);
- }
- **public static void** main(String[] args) {
- **new** Simple2();
- }}