

## The AWT package



(and SWING...)



## Abstract Windowing Toolkit

Java provides a package called

“Abstract Windowing Toolkit (AWT)” aimed at:

- developing *User-Friendly* graphical interfaces, built up using widget like buttons, element lists, areas for displaying objects and images,...
- Handling the interaction between the end-user and the program, relaying on an event-based programming model

Every action performed by the user

on the graphical interface generates an *event*, which must be handled by the program:

*if the user push that button,  
then this code must be executed...*

## GUI (Graphical User Interface)

- AWT - Abstract Windowing Toolkit
  - Also referred to as “Awful Windowing Toolkit” (in Java 1.0)
  - It defines event-based framework for writing GUIs.
- Swing
  - Rich set of easy-to-use, easy-to-understand GUI Components (Java Beans) to allow you creating a GUI that you can be satisfied with.

## AWT

- The large majority of objects used in building an interface derives from the class Component
- To build up a graphical interface (GUI), components are added to a special Component: a “Container”
- As a Container is also a Component, a Container can contain other Containers
- Each AWT component uses *native code* to make itself visualized on the screen
  - Whenever the application runs under MS Windows, buttons are ACTUALLY Windows’ buttons!
  - Whenever the application runs on a Unix machine exploiting Motif, buttons are ACTUALLY Motif’s buttons!



## Creating a GUI

Whenever it is created either an application equipped with a GUI,

1. The GUI is built up adding Components to the Container objects
2. The proper event handlers are coded, in order to react to the user's actions
3. The interface is displayed (automatically done for applets)

As a GUI is displayed, the interpreter starts a new thread waiting for an event to be fired

Whenever a button is pressed, the mouse is moved, etc., such a thread executes the proper code in the event handler which is associated to the current action.

Because of the presence of such an auxiliary thread, the `main()` method can terminate just after the interface display.



## Class Categories in Package AWT

Graphics Classes	classes that define colors, images, polygons, etc.
Components	GUI components such as <i>Button</i> , <i>Menu</i> , <i>List</i> , ...
Layout Managers	they control the layout of elements within their own container

The AWT package contains several sub-packages:

- `java.awt.color`
- `java.awt.datatransfer`
- `java.awt.dnd`
- `java.awt.event`
- `java.awt.font`
- `java.awt.geom`
- `java.awt.print`
- ...



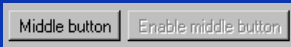
## Components

- Declaration:
- Creation:

```
Button b1;  
b1 = new Button(...);
```

Possible  
constructor  
parameters

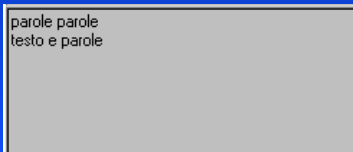
Button



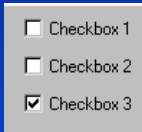
TextField



TextArea



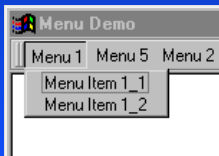
Checkbox



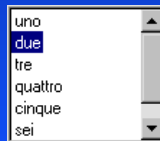
Choice



Menu



List



## Graphic Classes

java.lang.Object

Color

Dimension

Font

Graphics

Image

Point

Polygon

Rectangle

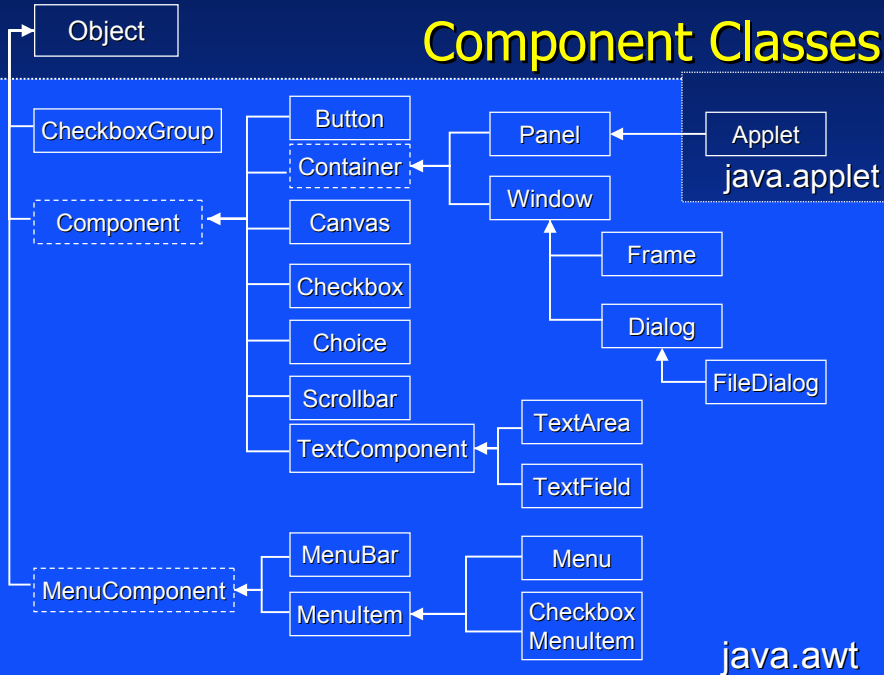
java.awt

Objects  
representing  
colors

Methods to  
draw texts,  
images, etc.

Classe

Classe  
Astratta



## Components

### Buttons

```

...
Button b=new Button("OK")
container.add(b);
...

```

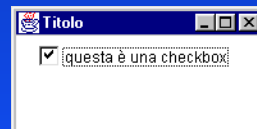


### Checkbox

```

...
Checkbox c=new Checkbox("questa è una checkbox");
container.add(c);
boolean stato=c.getState();
c.setState(nuovo stato);
...

```





## Components

### CheckboxGroup

```
CheckboxGroup cbg = new CheckboxGroup();
Checkbox cb1 =
    new Checkbox("Mangio la minestra", cbg, true);
Checkbox cb2 =
    new Checkbox("Mi butto dalla finestra", cbg, false);
container.add(cb1);
container.add(cb2);
```



### Label

```
Label lab1=new Label("Sono una label");
Label lab2=new Label("Anche io", Label.RIGHT);
container.add(lab1);
container.add(lab2);
```



## Components

### Choice

```
Choice c=new Choice();
c.add("Uno");
c.add("Due");
c.add("Tre");
container.add(c);
String s=c.getSelectedItem();
```

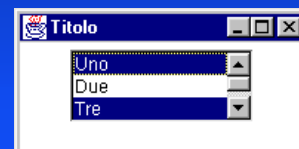


### List

```
List c=new List(3, true);
l.add("Uno");
l.add("Due");
l.add("Tre");
l.add("Quattro");
container.add(l);
String[] s=l.getSelectedItems();
```

# of elements  
to display

Multiple  
choices  
allowed





## Components

### TextField

#### Single-line text

```
TextField t=new TextField("Stringa iniziale");
t.setEditable(true);
String s=t.getText();
container.add(t);
```

Other methods:

- void setText(String text)
- void setEchoChar(char c)
- void select(int start, int end)
- String getSelectedText()



## Components

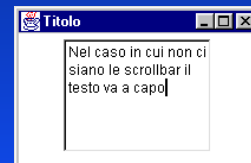
### TextArea

#### Multiple line text

```
TextArea t=new TextArea("Testo iniziale", 5, 15 );
t.setEditable(true);
String s=t.getText();
container.add(t);
```

Other methods and constructors:

- TextArea("Nel caso in cui non ci siano le "+  
"scrollbar il testo va a capo", 5, 15,  
TextArea.SCROLLBARS\_NONE);
- void setText(String text)
- void append(String s)
- void insert(String s, int pos)
- void select(int start, int end)
- String getSelectedText()





## Container: Frame

The objects **Panel**, **Window** (and their sub-classes **Frame** and **Applet**) are the only ones that can contain buttons, lists, etc.:  
the class **Container** is *abstract*  
**Frame** is a resizable window with a title

```
public class MyFrame {
    public static void main (String[] args ) {
        Frame fr = new Frame ("Titolo");
        fr.setSize(150,180);
        fr.setBackground( Color.white);
        fr.show();
    }
}
```

```
javac MyFrame.java
java MyFrame
```



Constructor  
for Frame

Methods of  
Frame, inherited  
from Component

The created object  
is not visible  
until the invocation  
of this method



## Container: Panel

Like **Window**, it provides space for other components  
To become visible, it must be placed onto a **Window** or **Frame**  
object (using method **add()** of class **Container**)

```
import java.awt.*;
public class MyFrame {
    public static void main (String[] args) {
        Frame fr = new Frame ("Titolo");
        Panel pnl = new Panel();
        fr.resize(200,200);
        fr.setBackground( Color.white);
        fr.setLayout(null);
        pnl.resize(100,100);
        pnl.setBackground( Color.cyan );
        fr.add(pnl);
        fr.show();
    }
}
```



Eliminates  
the default layout manager

Adds Panel  
onto Frame

Redraws the Frame  
and all the contained Components





## Layout Manager

Every Container (Panel, Frame, ...)  
is assigned a default **LayoutManager** object

A **LayoutManager** is in charge of placing components  
onto a **Container**  
according to its own rules.

The **LayoutManager** associated to a **Container**  
can be substituted by the programmer  
using the method **setLayout()** of class **Container**

```
Frame fr = new Frame ();  
fr.setLayout( lm );  
...  
fr.show();
```

Object of type  
**LayoutManager**



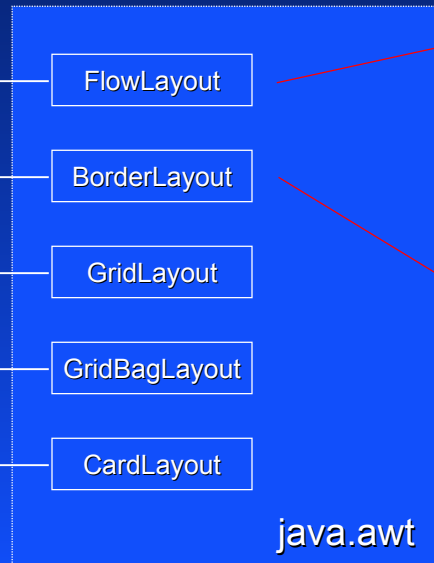
## Layout Managers

- **null** – Each component has specific place and size.
- **FlowLayout** – Components are added from left to right, top to bottom, by the order they are added to the container (components are sized according to preferred size)
- **GridLayout** – Table-like order.
- **BorderLayout** – 5 regions subdivision.
- **GridbagLayout** – The most robust and complicated one.
- **BoxLayout** – Order by *y* or *x* axis



Object

## Classes for Layout Managers



Default for  
Panel,  
Applet

Default for  
Window,  
Dialog,  
Frame

java.awt



## FlowLayout Manager (I)

Every object added to a **Container** with a **FlowLayout** manager follows the previous one, as in an horizontal line

The list of objects can be aligned (**LEFT**, **CENTER**, **RIGHT**), and the distance between components can be specified

```

Frame fr = new Frame ();
fr.setLayout( new FlowLayout() );
fr.setLayout( new FlowLayout(FlowLayout.RIGHT, 20, 40) );
fr.setLayout( new FlowLayout(FlowLayout.LEFT) );
...
fr.show();
  
```

Container

Creation of  
LayoutManager

alignment

Horizontal gap

Vertical gap



## FlowLayout Manager (II)

```
public class HorizFlow {
    public static void main (String[] args) {
        Frame fr = new Frame ("Titolo");
        fr.setLayout( new FlowLayout() );
        fr.add( new Button("OK") );
        fr.add( new Button("Annulla") );
        fr.add( new Button("Help") );
        fr.show();
    }
}
```



Resize  
with  
mouse



The components' size is not modified

Their position changes, according to the Frame size



## BorderLayout Manager (I)

It defines five areas on a Container:

North (up)    South (down)    East(right)    West (left)    Center



resize



The relative position of elements remains the same;  
but their size is modified



## BorderLayout Manager (II)

Code for the window in the previous slide:

```
public class Test {
    public static void main (String[] args) {
        Frame fr = new Frame ("Titolo");
        fr.add( "East", new Button("Destra") );
        fr.add( "North", new Button("Sopra") );
        fr.add( "West", new Button("Sinistra") );
        fr.add( "South", new Button("Sotto") );
        fr.add( "Center", new Button("Centro!") );
        fr.show();
    }
}
```

The invocation `fr.setLayout( new BorderLayout() );` is missing, as it is the default Layout Manager for Frame

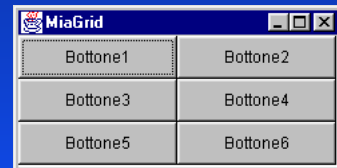


## GridLayout Manager

A GridLayout manager is created with a certain number of rows and columns

Components are added just filling a row after the other

```
public class Test {
    public static void main (String[] args) {
        Frame fr = new Frame ("MiaGrid");
        fr.setLayout( new GridLayout(3,2));
        for (int i=1; i<7; i++)
            fr.add(new Button("Bottone"+i) );
        fr.show();
    }
}
```



As for BorderLayout only the components' size is changed



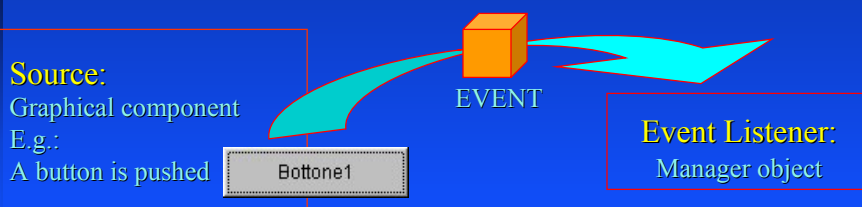
## Events (I)

- Graphical objects (Button ...) “fire events” as a consequence of the user’s actions:
  - A mouse click on an element;
  - A window resizing;
  - Selection of an element;
  - etc...
- Each event type is represented by a specific class
- Firing an event means creating an object (I.e. an instance of the class which describes the event)



## Events (II)

- The event, after its firing, is cached by one or more “listener”
- The listeners react to the event by executing some methods
- The point (in the code!) where an event takes place, and the location where it is handled can be separate





## Event Listener

It's an object of a class that implements a particular interface

```
public interface XXXListener {  
    public abstract void actionPerformed( XXXEvent e);  
}
```

```
class MioListener implements XXXListener {  
    public void actionPerformed( XXXEvent e) {  
        // code for managing the event  
    }  
}
```

This method  
is executed  
as the event is fired

Many interfaces of type XXXListener have the only method  
actionPerformed( XXXEvent )  
where XXX is the event name: Action, Mouse, ...



## "Registration" of an Event Listener

The "registration" is the operation to let the event producers know  
who they have to send a fired event to.

The registration is done by the methods addXXXListener()

The programmer is required to:

1. Define the graphical elements:  
    Button b1 = new Button("Ok");
2. Create the *event listener*  
    (an instance of the class that handles the event)  
    XXXListener ml = new XXXListener();
3. Register it to the object that produces such an event  
    b1.addXXXListener( ml );



## Event Management

```

Button b1;                                //Declaration of a graphical object
...
b1=new Button("ClickOnMe")                //Creation of the graphical object
                                           //not visible yet
MyButtListener mngr=new MyButtListener(); //Creation of an obj
                                           //able to handle the event types
                                           //I'm interested in (types are specified
                                           //by interfaces)

b1.addActionListener(mngr);               //Association of an event manager
                                           //to the event producer
  
```

When the user push the button:

- an instance of `ActionEvent` is created
- the method `actionPerformed()` is executed, with such an object as the parameter



## Event Listener: Use of *Inner Classes*

```

import java.awt.*;
import java.awt.event.*;
public class MyEvent {
    Frame fr; // visible from inside class MyEvent
    Button b1; // thus also from the INNER class MyButtListener
    public static void main(String[] args) {
        MyEvent e = new MyEvent();
        e.myOpen();
    }
    void myOpen() {
        fr = new Frame("My Event");
        fr.setLayout( new FlowLayout() );
        b1 = new Button("On");
        b1.addActionListener( new MyButtListener() );
        fr.add( b1 ); // add button to Frame
        fr.show();
    }
    class MyButtListener implements ActionListener {
        public void actionPerformed( ActionEvent e ) {
            if ( b1.getLabel().equals("On") ) b1.setLabel("Off");
            else b1.setLabel("On");
        }
    }
}
  
```

Add for the event management

Pushing the button, the label switches from On to Off and back.

Non-static access to the *InnerClass*

Creation and registration of EventListener

Event management



## EventListeners: Interfaces with Multiple Methods

- Interface ActionListener has the only method actionPerformed()
- Some other interfaces have more methods, in addition to actionPerformed()

*The classes that extend such interfaces must implement ALL the methods, regardless of their actual use*

- This can be boring. A possible trick is the use of *adapter classes*:
  - They implement such interfaces,
  - And define ALL the methods, but empty

The programmer can:

- Write an event listener as a class that extends the *adapter class*
- *override* only the useful methods



## Event Types and Syntax

All the AWT components have the methods:

- addXXXListener()
- removeXXXListener()

XXX represents the event type, and it is present in:

- The method name: addXXXListener()
- The name of the class describing the event: class XXXEvent
- The interface to be implemented by the listener class interface: XXXListener





## Events and Listeners (1)

Event types and their generator components

<u>event type</u>	<u>component</u>	<u>event description</u>
ActionEvent	Button, List, TextField, ...	button push, element selection in list, writing in TextField
AdjustmentEvent	Scrollbar, ...	scrollbar movement
ComponentEvent	Component and sub-cl	movement and resizing
KeyEvent	Component and sub-cl	key push
MouseEvent	Component and sub-cl	pointer movement, click, double click...



## Events and Listeners (2)

- Each component supports only some event types
- To know what the events associated to a component are, we can read the API documentation for that component:

e.g. Button has method `addActionListener()`

➔ Button can fire an ActionEvent

Button derives from Component

Component has methods:

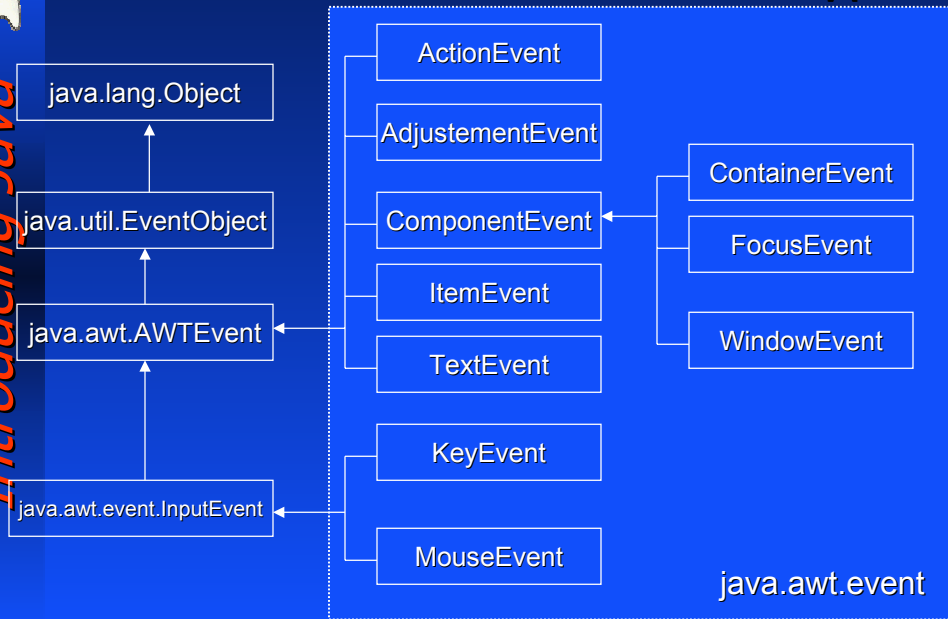
`addFocusListener,`  
`addKeyListener,`  
`addMouseListener, ...`

➔ Button can fire:

FocusEvent,  
KeyEvent,  
MouseEvent, ...



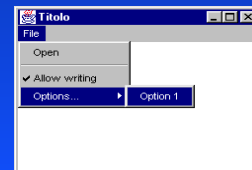
## Event Types



## Menu

Menus can be added to objects of class `Frame`, or its subclasses

1. Create a `MenuBar` object  
`MenuBar mb=new MenuBar();`
2. Create a `Menu` object  
`Menu m=new Menu("File");`
3. Create `MenuItem` objects, and add them to the menu  
`m.add(new MenuItem("Open"));`  
`m.addSeparator();` // add a separator  
`m.add(new CheckboxMenuItem("Allow writing"));` // Checkbox  
`Menu sub = new Menu("Options...");` //Create a submenu  
`sub.add(new MenuItem("Option 1"));`  
`m.add(sub);` // add submenu
4. Adds `Menu` to `MenuBar` (rightward)  
`mb.add(m);`
5. Add `MenuBar` to window by the proper method  
`frame.setMenuBar(mb);`





## Example 1

Create a window with three buttons, and handle the event fired as they are pushed.

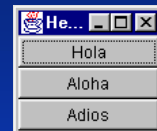
```
public class Activator {
    public static void main(String[] args) {
        Button b;
        ActionListener al = new MyActionListener();
        Frame f = new Frame("Hello Java");
        f.add(b = new Button("Hola"), BorderLayout.NORTH);
        b.setActionCommand("Hello");
        b.addActionListener(al);
        f.add(b = new Button("Aloha"), BorderLayout.CENTER);
        b.addActionListener(al);
        f.add(b = new Button("Adios"), BorderLayout.SOUTH);
        b.setActionCommand("Quit");
        b.addActionListener(al);
        f.pack();
        f.show();
    }
}
```

The command associated to buttons could be different to the one visualized on it  
(internationalization)



## Example 1

```
class MyActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        String s = e.getActionCommand();
        if (s.equals("Quit")) {
            System.exit(0);
        } else if (s.equals("Hello")) {
            System.out.println("Bon Jour");
        } else {
            System.out.println(s + " selected");
        }
    }
}
```



A different ActionListener can be associated to each button, so that we don't have to check what button generated the event (IDE)



```
public class Draw extends Frame{
    public void init() {
        addMouseListener(
            new MouseAdapter() {
                int savedX, savedY;
                public void mousePressed(MouseEvent e) {
                    savedX = e.getX();
                    savedY = e.getY();
                }
                public void mouseReleased(MouseEvent e) {
                    Graphics g = Draw.this.getGraphics();
                    g.drawRect(savedX, savedY,
                        e.getX()-savedX,
                        e.getY()-savedY);
                }
            }
        );
    }
}
```

## Example 2

Usare una classe adapter come classe interna anonima per disegnare rettangoli all'interno di una finestra: pressione del bottone per l'angolo in alto a sx, rilascio del bottone del mouse per l'angolo in basso a dx

```
public static void main (String[] args){
    Draw d=new Draw();
    d.init();
    d.show();
}
```



## Swing

- Il package swing estende le capacità grafiche di Java introducendo un nuovo insieme di componenti che affiancano quelli di AWT
- In molti casi i widget di swing sono analoghi a quelli di AWT, ma con una differenza sostanziale
  - I widget di AWT sono realizzati usando i widget nativi del sistema operativo sottostante (peer-based components)
  - I widget di swing possono esistere anche senza che esista un corrispondente nel sistema operativo sottostante (lightweight components)
- Per questo motivo una interfaccia swing ha un "look and feel" che è indipendente dal sistema operativo e può essere cambiato dinamicamente (il "look and feel" di default è detto *Metal*)
- Swing introduce anche alcuni nuovi componenti



## Swing: Some Components

- JPanel

è un oggetto simile a Panel (lightweight) con il supporto per il double-buffering

- Icon

non è un componente ma può essere usata con quasi tutti i componenti (è possibile inserire icone all'interno di bottoni ed altri componenti)

Oggetti che agiscono da icone implementano l'interfaccia Icon:

```
public interface Icon {  
    void paintIcon( Component c, Graphics g, int x, int y);  
    int getIconWidth();  
    int getIconHeight();  
}
```

La classe ImageIcon implementa l'interfaccia Icon e può essere usata per rappresentare immagini

```
Icon tinyPicture = new ImageIcon("TinyPicture.gif");
```



## Swing: Some Components

Esistono costruttori di ImageIcon che accettano come parametro un URL o un byte array

E' possibile creare icone implementando l'interfaccia ad es. nel seguente modo:

```
public class RedOval implements Icon {  
    public void paintIcon (Component c, Graphics g, int x, int y) {  
        g.setColor(Color.red);  
        g.drawOval (x, y, getIconWidth(), getIconHeight());  
    }  
    public int getIconWidth() { return 10; }  
    public int getIconHeight() { return 10; }  
}
```



## TextArea



## Swing: JTextPane

JTextPane is a text editor that supports text formatting, word-wrapping and visualization of pictures

1. Create a DefaultStyledDocument:  
DefaultStyledDocument doc=new DefaultStyledDocument();
2. Create a JTextPane passing the DefaultStyleDocument  
JTextPane pane=new JTextPane(doc);
3. Insert the JTextPane upon a JScrollPane:      JScrollPane  
scrollPane=new JScrollPane(pane);
4. Define styles and associate them to document portions



## Swing: JTextPane

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

public class Esempio2 {
    public static void main(String[] args) {
        JFrame f=new JFrame("Titolo");
        Container c=f.getContentPane();
        DefaultStyledDocument doc=new DefaultStyledDocument();
        JTextPane tp=new JTextPane(doc);
        c.add(new JScrollPane(tp));

        SimpleAttributeSet normal=new SimpleAttributeSet();

        SimpleAttributeSet italic=new SimpleAttributeSet();
        StyleConstants.setItalic(italic, true);

        SimpleAttributeSet big=new SimpleAttributeSet();
        StyleConstants.setFontSize(big, 36);
        ...
        doc.insertString(doc.getLength(), "Titolo\n", big);
        doc.insertString(doc.getLength(), "Corsivo\n", italic);
        doc.insertString(doc.getLength(), "Normale", normal);
        ...
        f.show();
    }
}
```

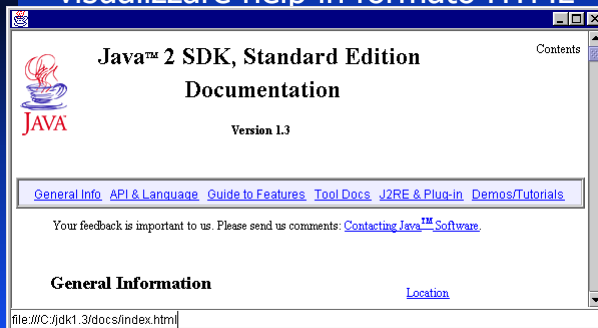




## Swing: JEditorPane

JEditorPane è un editor di testo che supporta testi in formato RTF e HTML

Non è un browser completo, ma può essere usato per visualizzare help in formato HTML



## Example: a Browser (1)

L'immagine del lucido precedente è stata realizzata con la seguente classe

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.text.*;
public class Browser extends JPanel{
    public Browser(){
        setLayout(new BorderLayout());
        final JEditorPane jt = new JEditorPane();
        final JTextField input = new JTextField("http://www.unipi.it");
        // Il JEditorPanel non è modificabile
        jt.setEditable(false);
        //CONTINUA
    }
}
```





## Example: a Browser (2)

```
// Per seguire i link
jt.addHyperlinkListener(new HyperlinkListener () {
    public void hyperlinkUpdate(final HyperlinkEvent e) {
        if (e.getEventType() == HyperlinkEvent.EventType.ACTIVATED) {

            // Salva il vecchio doc
            Document doc = jt.getDocument();
            try {
                URL url = e.getURL();
                jt.setPage(url);
                input.setText (url.toString());
            } catch (IOException io) {
                JOptionPane.showMessageDialog (
                    Browser.this, "Link non valido",
                    "Input non valido",
                    JOptionPane.ERROR_MESSAGE);
                jt.setDocument (doc);
            }
        }
    }
});
//CONTINUA
```



## Example: a Browser (3)

```
JScrollPane pane = new JScrollPane();
pane.getViewport().add(jt);
add(pane, BorderLayout.CENTER);

input.addActionListener (new ActionListener() {
    public void actionPerformed (ActionEvent e) {
        try {
            jt.setPage (input.getText());
        } catch (IOException ex) {
            JOptionPane.showMessageDialog (
                Browser.this, "URL non valido",
                "Input non valido",
                JOptionPane.ERROR_MESSAGE);
        }
    }
});
add (input, BorderLayout.SOUTH);
}
//CONTINUA
```

## Example: a Browser (4)

```
public static void main(String[] args) {  
    JFrame fr=new JFrame();  
    Container c=fr.getContentPane();  
  
    Browser b=new Browser();  
    c.add(b);  
    fr.show();  
}
```