

Syntax : void itemStateChanged(ItemEvent ie)

The keyListener interface

The interface defines three methods :

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

The MouseListener interface

This interface defines five methods.

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

The MouseMotionListener interface

This interface defines two methods.

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

The TextListener interface

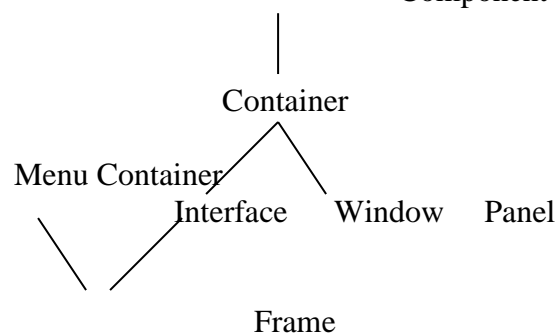
Syntax : void textChanged(TextEvent te)

AWT

AWT stands for **Abstract Window Toolkit**. The Abstract Windowing Toolkit (AWT) is Java's platform-independent windowing, graphics, and user-interface widget toolkit. The AWT is part of the Java Foundation Classes (JFC) - the standard API for providing a graphical user interface (GUI) for a Java program.

Window Fundamentals

The AWT defines windows according to a class hierarchy that adds functionality and specify with each level. The two most common windows are those derived from panel, which is used by applets, and those derived from Frame, which creates a standard window.



Component

At the top the AWT hierarchy is the Component class. Component is an abstract class that encapsulates all of the attributes of a visual component. All user interface elements that are displayed on the screen and that interact with the user are subclasses managing events, such as mouse and keyboard input, positioning and sizing the window, and repainting. A component object is responsible for

remembering the current foreground and background colors and the currently selected text font.

Container

The container class is a subclass of component. It has additional methods that allow other component objects to be nested within it. Other container objects can be stored inside of a Container. A Container is responsible for laying out (that is positioning) any components that it contains.

Panel

The Panel class is a concrete subclass of Container. It doesn't add new methods; it simply implements Container.

Window

The Window class creates a top-level window. A top-level window is not contained within any other object; it sits directly on the desktop. Generally, we won't create Window objects directly. Instead, we will use a subclass of Window called Frame.

Frame

Frame encapsulates what is commonly thought of as a "window". It is a subclass of Window and has a title bar, menu bar, borders, and resizing corners. When a Frame window is created by a program rather than an applet, a normal window is created.

Working with Frame Windows

After the applet, the type of window we will most often create is derived from Frame. We will use it to create child windows within applets, and top-level or child windows for applications.

There are two types of Frame constructors :

a) Frame() -> It creates a standard window that does not contain a title.

b) Frame(String title) -> It creates a window with the title specified by title. Here, we can't specify the dimensions of the window. Instead, we must set the size of the window after it has been created.

Setting the window's Dimensions

a) setSize() -> It is used to set the dimensions of the window.

Void setSize(int newWidth, int newHeight)

Void setSize(Dimension newSize)

The new size of the window is specified by newWidth and new Height, or by the width and height fields of the Dimension object passed in newSize. The dimensions are specified in terms of pixels.

Hiding and Showing a window

After a frame window has been created, it will not be visible until you call setVisible().

```
void setVisible(boolean visibleflag)
```

The component is visible if the argument to this method is true. Otherwise, it is hidden.

Setting a window's Title

We can change the title in a frame window using **setTitle()** method.

```
void setTitle(String newTitle)
```

Here, newTitle is the new title for the window.

Closing a Frame window

When using a frame window, our program must remove that window from the screen when it is closed, by calling **setVisible(false)**. To intercept a window-close event, we must implement the **windowClosing()** method of the **WindowListener** interface.

AWT Controls

Controls are components that allow a user to interact with your application and the AWT supports the following types of controls:

Labels, Push Buttons, Check Boxes, Choice Lists, Lists, Scrollbars, Text Components. These controls are subclasses of Component.

- 1) **Labels** : This is the simplest component of Java Abstract Window Toolkit. This component is generally used to show the text or string in your application and label never perform any type of action. Syntax for defining the label only and with justification :

```
Label label_name = new Label ("This is the label text.");
```

Above code simply represents the text for the label.

```
Label label_name = new Label ("This is the label text.", Label.CENTER);
```

Justification of label can be left, right or centered. Above declaration used the center justification of the label using the **Label.CENTER**.

- 2) **Buttons** : This is the component of Java Abstract Window Toolkit and is used to trigger actions and other events required for your application. The syntax of defining the button is as follows :

```
Button button_name = new Button ("This is the label of the button.");  
we can change the Button's label or get the label's text by using the  
Button.setLabel(String) and Button.getLabel() method. Buttons are added to the  
it's container using the add (button_name) method.
```

- 3) **Check Boxes** : This component of Java AWT allows you to create check boxes in your applications. The syntax of the definition of Checkbox is as follows :

```
Checkbox checkbox_name = new Checkbox ("Optional check box 1", false);
```

Above code constructs the unchecked Checkbox by passing the boolean valued argument *false* with the Checkbox label through the Checkbox() constructor. Defined Checkbox is added to it's container using add (checkbox_name) method. we can change and get the checkbox's label using the setLabel (String) and getLabel() method. we can also set and get the state of the checkbox using the setState(boolean) and getState() method provided by the **Checkbox** class.

- 4) **Radio Button** : This is the special case of the Checkbox component of Java AWT package. This is used as a group of checkboxes which group name is same. Only one Checkbox from a Checkbox Group can be selected at a time. Syntax for creating radio buttons is as follows :

```
CheckboxGroup chkgp = new CheckboxGroup();
```

```
add (new Checkbox ("One", chkgp, false);
```

```
add (new Checkbox ("Two", chkgp, false);
```

```
add (new Checkbox ("Three",chkgp, false);
```

In the above code we are making three check boxes with the label "One", "Two" and "Three". If you mention more than one true valued for checkboxes then your program takes the last true and show the last check box as checked.

- 5) **Text Area**: This is the text container component of Java AWT package. The Text Area contains plain text. TextArea can be declared as follows:

```
TextArea txtArea_name = new TextArea();
```

we can make the Text Area editable or not using the setEditable (boolean) method. If you pass the boolean valued argument *false* then the text area will be non-editable otherwise it will be editable. The text area is by default in editable mode. Text are set in the text area using the setText(string) method of the **TextArea** class.

- 6) **Text Field**: This is also the text container component of Java AWT package. This component contains single line and limited text information. This is declared as follows :

```
TextField txtfield = new TextField(20);
```

You can fix the number of columns in the text field by specifying the number in the constructor. In the above code we have fixed the number of columns to 20.

The Menu components

There are two types pop up and pull down. Pull down menus are accessed via a menu bar, which may contain multiple menus. Menu bars may appear only in frames.

To create a frame with a menu bar containing a pull down menu:

- Create a menu bar and attach it to frame.
- Create and populate the menu.
- Attach the menu to the menu bar.

To create a menu bar, instantiate `MenuBar` class. To attach it to frame, pass it into the frame's `setMenuBar()`.

To create a menu, instantiate `Menu` class. The most common constructor takes a string that is the menu's label. There are four kinds of elements that can be mixed and matched to populate a menu:

- **MenuItems**
 - A menu item is an ordinary textual component available on a menu. The basic constructor is `MenuItem(String text)` where `text` is the label of the menu item. A menu item is very much like a button that happens to live in a menu. Like buttons, menu items generates action events.
- **Checkbox MenuItems**
 - A checkbox menu item looks like a menu item with a checkbox to the left of its label. When a checkbox menu item is selected, the checkbox changes its state. The basic constructor is `CheckboxMenuItem(String label)`; You can read and set an item's state by calling `getState()` and `setState()`. These generate `ItemEvents`.
- **Separators**
 - A separator is just a horizontal mark used for visually dividing a menu into sections. To add a separator to a menu, call the menu's `addSeparator()` method().
- **Menus**
 - When you add a menu to another menu, the first menu's label appears in the second menu, with a pull-right icon. Pulling the mouse to the right causes the sub-menu to appear.

After a menu is fully populated, you attach it to a menu bar by calling menu bar's `add()` method. If you want the menu to appear in the Help menu position to the right of all other menus, call instead `setHelpMenu()`.

Menu Example

```
//example on Menu Components
//contains MenuBar,Menus and MenuItems
//menuexample.java

import java.awt.*;
import java.awt.event.*;

public class menuexample extends Frame implements ActionListener
{
    MenuBar mb;
    Menu m1,m2,m3;
    MenuItem mi1,mi2,mi3,mi4,mi5,mi6,mi7,mi8;
```

```

public menuexample()
{
    mb = new MenuBar();
    m1 = new Menu("File");
    m2 = new Menu("Edit");
    m3 = new Menu("Quit");

    mi1 = new MenuItem("New...");
    mi2 = new MenuItem("Open");
    mi3 = new MenuItem("Save");

    m1.add(mi1);
    m1.add(mi2);
    m1.add(mi3);

    mi4 = new MenuItem("cut");
    mi5 = new MenuItem("copy");
    mi6 = new MenuItem("paste");

    m2.add(mi4);
    m2.add(mi5);
    m2.add(mi6);

    mi7 = new MenuItem("Help");
    mi8 = new MenuItem("Exit");

    m3.add(mi7);
    m3.add(mi8);

    mb.add(m1);
    mb.add(m2);
    mb.add(m3);

    setMenuBar(mb);

    setTitle("menu example....");
    setVisible(true);
    setSize(300,300);
    addWindowListener(new WindowAdapter()
    {
        public void windowClosing(WindowEvent we)
        {
            System.exit(0);
        }
    });
    mi8.addActionListener(this);
} //constructor
public void actionPerformed(ActionEvent ae)
{
    System.exit(0);
}

```

```
    }  
    public static void main(String[] args)  
    {  
        menuexample me = new menuexample();  
    }  
}
```

What is a Layout Manager?

A layout manager is an object that controls the size and position (layout) of components inside a Container object. For example, a window is a container that contains components such as buttons and labels. The layout manager in effect for the window determines how the components are sized and positioned inside the window.

List of Layout Managers

The java.awt package provides the following predefined layout managers that implement the java.awt.LayoutManager interface. Every Abstract Window Toolkit (AWT) and Swing container has a predefined layout manager as its default. It is easy to use the container.setLayout method to change the layout manager, and you can define your own layout manager by implementing the java.awt.LayoutManager interface. This article describes the predefined AWT layout managers in this list.

[java.awt.BorderLayout](#)
[java.awt.FlowLayout](#)
[java.awt.CardLayout](#)
[java.awt.GridLayout](#)
[java.awt.GridBagLayout](#)

BorderLayout

The border layout is the default layout manager for all Window objects. It consists of five fixed areas: North, South, East, West, and Center. You do not have to put a component in every area of the border layout. The demonstration puts the label "Border Layout" in the North area, the OK button in the South area, and the List of fruit trees in the Center area. The East (right) and West (left) areas are empty.

FlowLayout

The flow layout is the default layout manager for all Panel objects and applets. It places the panel components in rows according to the width of the panel and the number and size of the components. FlowLayout implements a simple layout style, which is similar to how words flow in a text editor. Components are laid out from the upper-left corner, left to right and top to bottom. When no more components fit on a line, the next one appears on the next line.

GridLayout

The Grid layout arranges components into a grid of rows and columns. we specify the number of rows and columns, the number of rows only and let the layout manager determine the number of columns, or the number of columns only and let the layout manager determine the number of rows.