

# Java's Basic User Interface Components

By Gayathri Gokul

2003-10-20

Printed from DevShed.com

URL: [http://www.devshed.com/Server\\_Side/Java/Java\\_Basic\\_User\\_Interface\\_Components/](http://www.devshed.com/Server_Side/Java/Java_Basic_User_Interface_Components/)

## Java's Basic User Interface Components

To simplify user interaction and make data entry easier, you can use java controls. Controls are components, such as buttons, labels and text boxes, that can be added to containers like frames, panels and applets. The Java.awt package provides an integrated set of classes to manage user interface components.

Components are placed on the user interface by adding them to a container. As we saw in the previous tutorial, a container itself is a component. The easiest way to demonstrate interface design is by using the container you have been working all along with – the Applet class. The simplest form of Java AWT component is the basic User Interface Component. You can create and add these to your applet without needing to know anything about creating containers or panels; in fact, your applet, even before you start painting and drawing and handling events, is already an AWT container. Because an applet is a container, you can put other AWT components, such as UI components or other containers, in it.

In this tutorial, you'll learn about the basic UI components (controls) like labels, buttons, check boxes, choice menus, and text fields. The classes involved and take a look at each component and its function with coded illustrations. The following table gives a list of all the Controls in Java AWT and their respective functions.

### Adding a Control to a Container

In order to add a control to a container, you need to perform the following two steps.

- Create an object of the control by passing the required arguments to the constructor.
- Add the component (control) to the container.

### Controls in Java

| Controls     | Functions                                                                                                           |
|--------------|---------------------------------------------------------------------------------------------------------------------|
| Textbox      | Accepts single line alphanumeric entry.                                                                             |
| TextArea     | Accepts multiple line alphanumeric entry.                                                                           |
| Push button  | Triggers a sequence of actions.                                                                                     |
| Label        | Displays Text.                                                                                                      |
| Check box    | Accepts data that has a yes/no value. More than one checkbox can be selected.                                       |
| Radio button | Similar to check box except that it allows the user to select a single option from a group.                         |
| Combo box    | Displays a drop-down list for single item selection. It allows new value to be entered.                             |
| List box     | Similar to combo box except that it allows a user to select single or multiple items. New values cannot be entered. |

### Classes Involved

| Controls     | Class                       |
|--------------|-----------------------------|
| Textbox      | TextField                   |
| TextArea     | TextArea                    |
| Push button  | Button                      |
| Check box    | CheckBox                    |
| Radio button | CheckboxGroup with CheckBox |
| Combo box    | Choice                      |
| List box     | List                        |

## The Button Class

We will start with the simplest of UI components: the button. Buttons are used to trigger events in a GUI environment (we have discussed Event Handling in detail in the previous tutorials). They are easy to manage and, most importantly, they are easy to use. The Button class is used to create buttons. When you add components to the container, you don't specify a set of coordinates that indicates where the components are to be placed. The arrangement of components is handled by a layout manager in effect for the container. The default layout for a container is flow layout. Now let us write a simple code to test our button class.

To create a button use, one of the following constructors:

- Button() creates a button with no text label.
- Button(String) creates a button with the given string as label.

### Example 1

```
/*
<Applet code= "ButtonTest.class"
width = 500
height = 100>
</applet>
*/

import java.awt.*;
public class ButtonTest extends java.applet.Applet
{
    Button b1 = new Button ("Play");
    Button b2 = new Button ("Stop");

    public void init(){
        add(b1);
        add(b2);
    }
}
```

The following Java application illustrates addition of button controls to a panel.

### Example 2

```
import java.awt.*;

public class ButttonTest2 extends Frame
{
    public static void main (String arg[])
    {
        Frame frm;
        Panel panel,
        Button b1, b2,
        frm= new Frame("Using Frame As Container For-Button Test");
        frm.setSize(300,400);
        frm.setBackground (Color.green);
        frm.setVisible(true)

        panel = new Panel();
        b1 = new Button("Accept");
        b2 = new Button("Cancel");

        frm.add(panel);
```

```

panel.add(b1);
panel.add(b2);
}
}

```

We can use the `setLabel()` method to change the label of the button and the `getLabel()` method to retrieve the caption.

### The Label Class

Labels are created via the `Label` class. Labels are often used to identify the purpose of other components on a given interface; they cannot be edited directly by the user. Using a label is much easier than using a `drawString()` method because labels are drawn automatically and don't have to be handled explicitly in the `paint()` method. Labels can be laid out according to the layout manager, instead of using `[x, y]` coordinates, as in `drawString()`.

To create a `Label`, use one of the following constructors:

- `Label()` – creates a label with its string aligned to the left.
- `Label(String)` – creates a label initialized with the given string, and aligned left.
- `Label(String, int)` – creates a label with specified text and alignment indicated by the `int` argument: `Label.Right`, `Label.Left` and `Label.Center`.

### Example 3

The following is a simple applet that creates a few labels in Helvetica bold.

```

/*
<Applet code= "LabelTest.class"
Width = 500
Height = 100>
</applet>
*/

import java.awt.*;

public class LabelTest extends java.applet.Applet
{
    Label left = new Label ("Left Wing");
    Label center = new Label ("Central Part", Label.CENTER);
    Label right = new Label ("Right Wing", Label.RIGHT);
    Font f1 = new Font("Helvetica", Font.Bold, 14);
    GridLayout gd = new GridLayout(3,1);

    public void init(){
        setFont(f1);
        setLayout(gd);
        add(left);
        add(center);
        add(right);
    }
}

```

We can use label's `getText()` method to indicate the current label's text and `setText()` method to change the label's text. We have also made use of `setFont()` method in the above example to change the label's font and `GridLayout` to lay the components in the applet.

### The Checkbox Class

Check Boxes are labeled or unlabeled boxes that can be either “Checked off” or “Empty”. Typically, they are used to select or deselect an option in a program, such as the “Disable sound” check boxes from a Windows screen.

Check Boxes are generally *nonexclusive*, which means that if you have six check boxes in container, all the six can either be checked or unchecked at the same time. This component can be organized into check box group, which is sometimes called radio buttons. Both kinds of check boxes are created using the `Checkbox` class. To create a nonexclusive check box you can use one of the following constructors:

- `Checkbox()` creates an unlabeled checkbox that is not checked.
- `Checkbox(String)` creates an unchecked checkbox with the given label as its string.

After you create a checkbox object, you can use the `setState(boolean)` method with a true value as argument for checked checkboxes, and false to uncheck it. Six checkboxes are created in **Example 4**, which is an applet to enable you to select up to six courses at a time. All five checkboxes are unchecked only the second option is checked.

Example 4

```
/*
<Applet code= "CheckboxTest.class"
Width = 200
Height = 150>
</applet>
*/

import java.awt.*;

public class CheckboxTest extends java.applet.Applet
{
    Checkbox c1 = new Checkbox ("Java");
    Checkbox c2 = new Checkbox ("XML");
    Checkbox c3 = new Checkbox ("VB");
    Checkbox c4 = new Checkbox ("Oracle");
    Checkbox c5 = new Checkbox ("SQL");
    Checkbox c6 = new Checkbox ("ASP");

    public void init(){
        Add(c1);
        c2.setStatus(true);
        add(c2);
        add(c3);
        add(c4);
        add(c5);
        add(c6);
    }
}
```

### The CheckboxGroup Class

`CheckboxGroup` is also called like a radio button or exclusive check boxes. To organize several `Checkboxes` into a group so that only one can be selected at a time, a `CheckboxGroup` object is created with the statement such as follows:

```
CheckboxGroup radio = new CheckboxGroup ();
```

The `CheckboxGroup` keeps track of all the check boxes in its group. We have to use this object as an extra argument to the ***Checkbox*** constructor.

`Checkbox (String, CheckboxGroup, Boolean)` creates a checkbox labeled with the given string that belongs to the `CheckboxGroup` indicated in the second argument. The last argument equals true if box is checked, false otherwise.

The following is an applet to enable you to select one of the ways to connect to the Internet.

Example 5

```
/*
<Applet code= "ChkGroup.class"
Width = 200
Height = 150>
</applet>
*/

Import java.awt.*;

Public class ChkGroup extends java.applet.Applet
{
CheckboxGroup cbgr = new CheckboxGroup();
Checkbox c1 = new Checkbox ("America Online", cbgr, false);
Checkbox c2 = new Checkbox ("MSN", cbgr, false);
Checkbox c3 = new Checkbox ("NetZero", cbgr, false);
Checkbox c4 = new Checkbox ("EarthLink", cbgr, false);
Checkbox c5 = new Checkbox ("Bellsouth DSL", cbgr, true);

Public void init(){
add(c1);
add(c2);
add(c3);
add(c4);
add(c5);
}
}
```

The setCurrent(checkbox) method can be used to make the set of currently selected check boxes in the group. There is also a getCurrent() method, which returns the currently selected checkbox.

### The Choice List Class

Choice Lists, which is created from the Choice class, are components that enable a single item to be picked from a pull-down list. We encounter this control very often on the web when filling out forms. The first step in creating a Choice List is to create a choice object to hold the list, as shown below:

```
Choice cgender = new Choice();
```

Items are added to the Choice List by using addItem(String) method on the object. The following code adds two items to the gender choice list.

```
cgender.addItem("Female"); cgender.addItem("Male");
```

After you add the Choice List it is added to the container like any other component using the add() method. The following example shows an Applet that contains a list of shopping items in the store.

```
/*
<Applet code= "ChoiceTest.class"
Width = 200
Height = 150>
</applet>
*/

Import java.awt.*;

Public class ChoiceTest extends java.applet.Applet
{
Choice shoplist = new Choice();
```

```

Public void init(){
shoplist.addItem("Bed And Bath");
shoplist.addItem("Furniture");
shoplist.addItem("Clothing");
shoplist.addItem("Home Appliance");
shoplist.addItem("Toys and Accessories");

add(shoplist);
}
}

```

The choice list class has several methods, which are given below:

| Method             | Action                                                                                             |
|--------------------|----------------------------------------------------------------------------------------------------|
| getItem()          | Returns the string item at the given position (items inside a choice begin at 0, just like arrays) |
| countItems()       | Returns the number of items in the menu                                                            |
| getSelectedIndex() | Returns the index position of the item that's selected                                             |
| getSelectedItem()  | Returns the currently selected item as a string                                                    |
| select(int)        | Selects the item at the given position                                                             |
| select(String)     | Selects the item with the given string                                                             |

### The TextField and TextArea Class

To accept textual data from a user, AWT provided two classes, **TextField** and **TextArea**. The TextField handles a single line of text and does not have scrollbars, whereas the TextArea class handles multiple lines of text. Both the classes are derived from the TextComponent class. Hence share many common methods. TextFields are different from labels in that they can be edited; labels are good for just displaying text, while for getting text input from the user, the best option is TextField.

**TextFields** provide an area where you can enter and edit a single line of text. To create a text field, use one of the following constructors:

- TextField() creates an empty TextField with no specified width.
- TextField(int) creates an empty text field with enough width to display the specified number of characters (this has been depreciated in Java2).
- TextField(String) creates a text field initialized with the given string.
- TextField(String, int) creates a text field with specified text and specified width.

For example, the following line creates a text field 25 characters wide with the string "Brewing Java" as its initial contents:

```
TextField txtfld = new TextField ("Brewing Java", 25); add(txtfld);
```

### Setting a Password Character

The text Field allows you to specify an echo character that helps you accept a password without displaying what the user has keyed in. For example, say you have specified "\*" as the echo character, and the user type in five character, "\*\*\*\*\*" is displayed in the text filed instead of the text the user has keyed in. To do this, first create the text field itself; then use the setEchoCharacter() method to set the character that is echoed on the screen. Here is an example:

```
TextField tf = new TextField(30); tf.setEchoCharacter('*');
```

The Applet in **Example7** below creates several text fields. Labels are used to identify the fields. One of the fields uses an obscuring character to hide text being input. This applet uses the default layout manager, the only thing causing the six components to appear in three different lines is the width of the window.

```

/*
<Applet code= "TextfieldTest.class"
Width = 400
Height = 150>
</Applet>
*/

Import java.awt.*;
Public class TextfieldTest extends java.applet.Applet
{
Label uLabel = new label ("User Name:");
TextField uText = new TextField (25);
Label eLabel = new label ("Email ID:");
TextField eText = new TextField (25);
Label pLabel = new label ("Admin Password:");
TextField pText = new TextField(25);

Public void init(){
add(uLabel);
add(uText);
add(eLabel);
add(eText);
add(pLabel);
pText.setEchoCharacter('*');
add(pText);
}
}

```

After creating the controls, we initialize them using the "new" keyword. Use the add() method to add controls to the container, in this case the applet. The other methods for TextField are:

| Method                        | Action                                                                      |
|-------------------------------|-----------------------------------------------------------------------------|
| String getText()              | Extracts the text from the text field (as a string).                        |
| Void setText(String str)      | Sets the text of the text field to the text specified in the str.           |
| Int getColumns()              | Returns the width of this text field.                                       |
| select(int, int)              | Selects the text between the two integer positions (positions start from 0) |
| selectAll()                   | Selects all the text in the field                                           |
| Boolean isEditable ()         | Returns true or false based on whether the text is editable                 |
| Char getEchoChar()            | Returns the character used for masking input                                |
| Boolean echoCharIsSet(char c) | Returns true or false based on whether the field has a masking character    |

### TextArea

The TextArea is an editable text field that can handle more than one line of input. Text areas have horizontal and vertical scrollbars to scroll through the text. Adding a text area to a container is similar to adding a text field. To create a text area use one of the following constructors:

- TextArea() creates an empty text area with unspecified width and height.
- TextArea(int, int) creates an empty text area with indicated number of lines and specified width in characters.
- TextArea(String) creates a text area initialized with the given string.
- TextField(String, int, int) creates a text area containing the indicated text and specified number of lines and width in the characters.

The Applet in **Example 8** displays a text area that is filled with a string, when the programs begins running using appletviewer.

```
/*
<Applet code= "TextAreaTest.class"
Width = 250
Height = 450>
</Applet>
*/

Import java.awt.*;
Public class TextfieldTest extends java.applet.Applet
{
String letter = "Dear Readers: \n" +
"We are learning about Java AWT. \n" +
"Some of you might find it all exciting, while a few are still not very sure \n" +
"For those cat on the wall folks \n" +
"My advice is read it again and practice some codes. \n \n" +
"So brewing a cup of Java? Isn't so hard as it looked...is it! " ;

TextArea ltArea;

Public void init(){
ltArea = new TextArea(letter, 10, 50)
add(ltArea);
}
}
```

The TextArea, like TextField, can use methods like setText(), getText(), setEditable(), and isEditable(). In addition, there are two more methods like these. The first is the insertText(String, int) method, used to insert indicated strings at the character index specified. The second is replaceText(String, int, int), used to replace text between given integer position with the indicated string.

It's this sort of real work in Java applets and applications for which Java's Abstract Windowing Toolkit, or AWT, was designed. You've actually been using the AWT all along, as you might have guessed from the classes you've been importing. The basic idea behind the AWT is that a graphical Java program is a set of nested components, starting from the outermost window all the way down to the smallest UI component. Components can include things you can actually see on the screen, such as windows, menu bars, buttons, and text fields, and they can also include containers, which in turn can contain other components. Hope you have got a clear picture of Java AWT and all the basic UI components, in the next part of the tutorial we will deal with more advance user interface components.