## 1. COMPONENT EVENT

Component events are provided for notification purposes ONLY; The AWT will automatically handle component moves and resizes internally so that GUI layout works properly regardless of whether a program is receiving these events or not. In addition to serving as the base class for other component-related events (InputEvent, FocusEvent, WindowEvent, ContainerEvent), this class defines the events that indicate changes in a component's size, position, or visibility.

This low-level event is generated by a component object (such as a List) when the component is moved, resized, rendered invisible, or made visible again. The event is passed to every ComponentListener or ComponentAdapter object which registered to receive such events using the component's addComponentListener method. (ComponentAdapter objects implement the ComponentListener interface.) Each such listener object gets this ComponentEvent when the event occurs.

**Declaration:**

```
public class ComponentEvent    extends AWTEvent
```

**Fields:**

Following are the fields for java.awt.Component class:

- static int COMPONENT_FIRST -- The first number in the range of ids used for component events.
- static int COMPONENT_HIDDEN --This event indicates that the component was rendered invisible.
- static int COMPONENT_LAST -- The last number in the range of ids used for component events.
- static int COMPONENT_MOVED -- This event indicates that the component's position changed.
- static int COMPONENT_RESIZED -- This event indicates that the component's size changed.

- static int COMPONENT_SHOWN -- This event indicates that the component was made visible.

**Class constructors**

- ComponentEvent(Component source, int id)- Constructs a ComponentEvent object.
- Component getComponent()- Returns the originator of the event.
- String paramString()- Returns a parameter string identifying this event.

**Component listener**

The class which processes the ComponentEvent should implement this interface.The object of that class must be registered with a component. The object can be registered using the addComponentListener() method. Component event are raised for information only.

**Declaration:**

```
public interface ComponentListener    extends EventListener
```

**Interface methods**

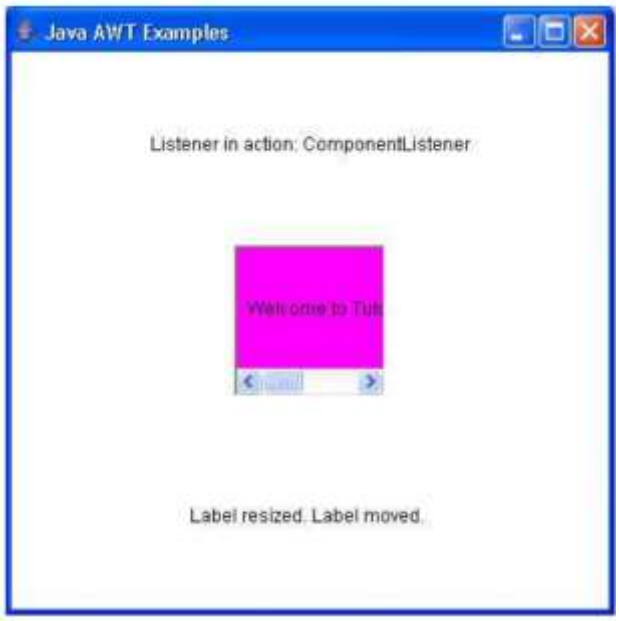
- void componentHidden(ComponentEvent e)- Invoked when the component has been made invisible.
- void componentMoved(ComponentEvent e)- Invoked when the component's position changes.
- void componentResized(ComponentEvent e)- Invoked when the component's size changes.
- void componentShown(ComponentEvent e)-Invoked when the component has been made visible.

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


public class AwtListenerDemo {
```

```
private Frame mainFrame;
private Label headerLabel;
private Label statusLabel;
private Panel controlPanel;

public AwtListenerDemo(){
  prepareGUI();
}

public static void main(String[] args){
  AwtListenerDemo  awtListenerDemo = new AwtListenerDemo();
  awtListenerDemo.showComponentListenerDemo();
}

private void prepareGUI(){
  mainFrame = new Frame("Java AWT Examples");
  mainFrame.setSize(400,400);
  mainFrame.setLayout(new GridLayout(3, 1));
  mainFrame.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent windowEvent){
      System.exit(0);
    }
  });

  headerLabel = new Label();
  headerLabel.setAlignment(Label.CENTER);
  statusLabel = new Label();
  statusLabel.setAlignment(Label.CENTER);
  statusLabel.setSize(350,100);

  controlPanel = new Panel();
  controlPanel.setLayout(new FlowLayout());
```

```
    mainFrame.add(headerLabel);
    mainFrame.add(controlPanel);
    mainFrame.add(statusLabel);
    mainFrame.setVisible(true);
  }

  private void showComponentListenerDemo(){
    headerLabel.setText("Listener in action: ComponentListener");

    ScrollPane panel = new ScrollPane();
    panel.setBackground(Color.magenta);

    Label msglabel = new Label();
    msglabel.setAlignment(Label.CENTER);
    msglabel.setText("Welcome to TutorialsPoint AWT Tutorial.");
    panel.add(msglabel);

    msglabel.addComponentListener(new CustomComponentListener());
    controlPanel.add(panel);
    mainFrame.setVisible(true);
  }

  class CustomComponentListener implements ComponentListener {

    public void componentResized(ComponentEvent e) {
      statusLabel.setText(statusLabel.getText()
      + e.getComponent().getClass().getSimpleName() + " resized. ");
    }

    public void componentMoved(ComponentEvent e) {
      statusLabel.setText(statusLabel.getText()
```

```
        + e.getComponent().getClass().getSimpleName() + " moved. ");
    }

    public void componentShown(ComponentEvent e) {
      statusLabel.setText(statusLabel.getText()
      + e.getComponent().getClass().getSimpleName() + " shown. ");
    }

    public void componentHidden(ComponentEvent e) {
      statusLabel.setText(statusLabel.getText()
      + e.getComponent().getClass().getSimpleName() + " hidden. ");
    }
  }
}
```




## 2. ADJUSTMENT LISTENER

The interfaceAdjustmentListener is used for receiving adjustment events. The class that process adjustment events needs to implements this interface.

**Declaration**

```
public interface AdjustmentListener extends EventListener
```

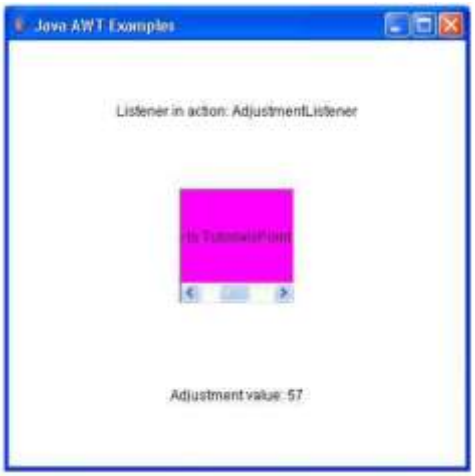
**Interface methods**

void adjustmentValueChanged(AdjustmentEvent e)- Invoked when the value of the adjustable has changed.

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

public class AwtListenerDemo {
  private Frame mainFrame;
  private Label headerLabel;
  private Label statusLabel;
  private Panel controlPanel;

  public AwtListenerDemo(){
    prepareGUI();
  }

  public static void main(String[] args){
    AwtListenerDemo  awtListenerDemo = new AwtListenerDemo();
    awtListenerDemo.showAdjustmentListenerDemo();
  }

  private void prepareGUI(){
    mainFrame = new Frame("Java AWT Examples");
    mainFrame.setSize(400,400);
    mainFrame.setLayout(new GridLayout(3, 1));
    mainFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent windowEvent){
        System.exit(0);
      }
```

```
      });

      headerLabel = new Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);

      controlPanel = new Panel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showAdjustmentListenerDemo(){
      headerLabel.setText("Listener in action: AdjustmentListener");

      ScrollPane panel = new ScrollPane();
      panel.setBackground(Color.magenta);
      panel.getHAdjustable().addAdjustmentListener(new
CustomAdjustmentListener());

      Label msglabel = new Label();
      msglabel.setAlignment(Label.CENTER);
      msglabel.setText("Welcome to TutorialsPoint AWT Tutorial.");
      panel.add(msglabel);

      controlPanel.add(panel);
      mainFrame.setVisible(true);
```

```
  }

  class CustomAdjustmentListener implements AdjustmentListener {
    public void adjustmentValueChanged(AdjustmentEvent e) {
      statusLabel.setText("Adjustment value: "+Integer.toString(e.getValue()));
    }
  }
}
```




### 3. CONTAINER LISTENER

The listener interface for receiving container events. The class that is interested in processing a container event either implements this interface (and all the methods it contains) or extends the abstract ContainerAdapter class (overriding only the methods of interest). The listener object created from that class is then registered with a component using the component's addContainerListener method. When the container's contents change because a component has been added or removed, the relevant method in the listener object is invoked, and the ContainerEvent is passed to it. Container events are provided for notification purposes ONLY; The AWT will automatically handle add and remove operations internally so the program works properly regardless of whether the program registers a ComponentListener or not.

**Interface methods**

- void componentAdded(ContainerEvent e)- Invoked when a component has been added to the container.
- void componentRemoved(ContainerEvent e)-Invoked when a component has been removed from the container.

4. **MOUSE LISTENER**

MouseListener and MouseMotionListener is an interface in java.awt.event package . Mouse events are of two types. MouseListener handles the events when the mouse is not in motion. While MouseMotionListener handles the events when mouse is in motion.There are five types of events that MouseListener can generate. There are five abstract functions that represent these five events. The abstract functions are :

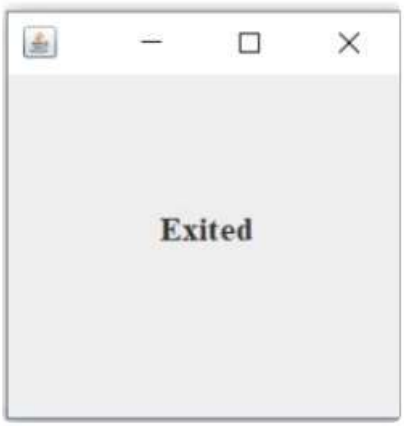
- void mouseReleased(MouseEvent e) : Mouse key is released
- void mouseClicked(MouseEvent e) : Mouse key is pressed/released
- void mouseExited(MouseEvent e) : Mouse exited the component
- void mouseEntered(MouseEvent e) : Mouse entered the component
- void mousepressed(MouseEvent e) : Mouse key is pressed

There are two types of events that MouseMotionListener can generate. There are two abstract functions that represent these five events. The abstract functions are :

- void mouseDragged(MouseEvent e) : Invoked when a mouse button is pressed in the component and dragged. Events are passed until the user releases the mouse button.
- void mouseMoved(MouseEvent e) : invoked when the mouse cursor is moved from one point to another within the component, without pressing any mouse buttons.

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

public class MyMouse extends JFrame implements MouseListener
{
```

```
  JLabel label;
  MyMouse(){
    addMouseListener(this);
    label = new JLabel();
    label.setBounds(90,80,130,20);
    label.setFont(new Font("Serif", Font.BOLD, 20));
    add(label);
    setSize(250,250);
    setLayout(null);
    setVisible(true);
  }
  public void mouseClicked(MouseEvent e) {
    label.setText("Clicked");
  }
  public void mouseEntered(MouseEvent e) {
    label.setText("Entered");
  }
  public void mouseExited(MouseEvent e) {
    label.setText("Exited");
  }
  public void mousePressed(MouseEvent e) {
    label.setText("Pressed");
  }
  public void mouseReleased(MouseEvent e) {
    label.setText("Released");
  }
  public static void main(String[] args) {
    new MyMouse();
  }
}
```




## 5. AWT CONTROLS

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. Every user interface considers the following three main aspects:

- UI elements : Thes are the core visual elements the user eventually sees and interacts with. GWT provides a huge list of widely used and common elements varying from basic to complex.
- Layouts: They define how UI elements should be organized on the screen and provide a final look and feel to the GUI (Graphical User Interface).
- Behavior: These are events which occur when the user interacts with UI elements.

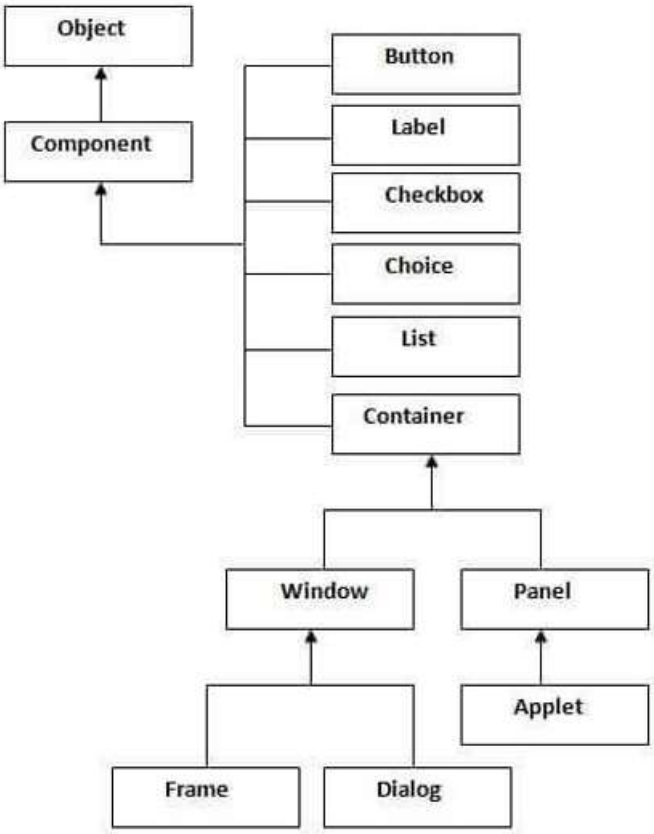



**Fig 1: Hierarchy of AWT controls**

- 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.

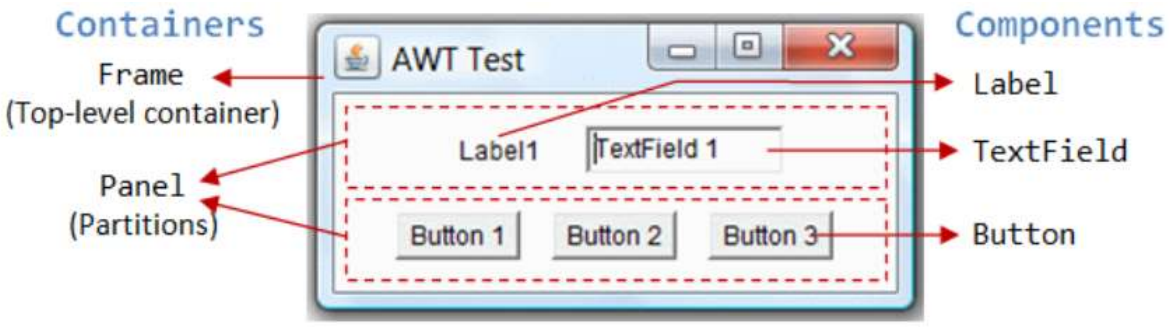



**Fig 2: AWT controls**

**UI elements:**

- Label-A Label object is a component for placing text in a container.
- Button-This class creates a labeled button.
- Check Box- A check box is a graphical component that can be in either an on (true) or off (false) state.
- Check Box Group-The CheckboxGroup class is used to group the set of checkbox.
- List- The List component presents the user with a scrolling list of text items.
- Text Field-A TextField object is a text component that allows for the editing of a single line of text.
- Text Area-A TextArea object is a text component that allows for the editing of a multiple lines of text.
- Choice-A Choice control is used to show pop up menu of choices. Selected choice is shown on the top of the menu.
- Canvas-A Canvas control represents a rectangular area where application can draw something or can receive inputs created by user.
- Image-An Image control is superclass for all image classes representing graphical images.
- Scroll Bar-A Scrollbar control represents a scroll bar component in order to enable user to select from range of values.
- Dialog-A Dialog control represents a top-level window with a title and a border used to take some form of input from the user.
- File Dialog-A FileDialog control represents a dialog window from which the user can select a file.

## 6. LABEL CONTROL

Label is a passive control because it does not create any event when accessed by the user. The label control is an object of Label. A label displays a single line of read-only text. However the text can be changed by the application programmer but cannot be changed by the end user in any way.

**Declaration**

```
public class Label extends Component  implements Accessible
```

**Class constructors**

- Label()- Constructs an empty label.
- Label(String text)- Constructs a new label with the specified string of text, left justified.
- Label(String text, int alignment)- Constructs a new label that presents the specified string of text with the specified alignment.

**Class methods**

- void addNotify()- Creates the peer for this label.
- AccessibleContext getAccessibleContext()- Gets the AccessibleContext associated with this Label.
- int getAlignment()- Gets the current alignment of this label.
- String getText()- Gets the text of this label.
- protected String paramString()-Returns a string representing the state of this Label.
- void setAlignment(int alignment)- Sets the alignment for this label to the specified alignment.
- void setText(String text)- Sets the text for this label to the specified text.

## 7. BUTTONS

A button is basically a control component with a label that generates an event when pushed. The Button class is used to create a labeled button that has platform

independent implementation. The application result in some action when the button is pushed. When we press a button and release it, AWT sends an instance of ActionEvent to that button by calling processEvent on the button. The processEvent method of the button receives the all the events, then it passes an action event by calling its own method processActionEvent. This method passes the action event on to action listeners that are interested in the action events generated by the button.

To perform an action on a button being pressed and released, the ActionListener interface needs to be implemented. The registered new listener can receive events from the button by calling addActionListener method of the button. The Java application can use the button's action command as a messaging protocol.

**Declaration**

```
public class Button extends Component implements Accessible
```

**Button Class Constructors**

- Button( )- It constructs a new button with an empty string i.e. it has no label.
- Button (String text)- It constructs a new button with given string as its label.

**Button Class Methods**

- void setText (String text)-It sets the string message on the button
- String getText()- It fetches the String message on the button.
- void setLabel (String label)- It sets the label of button with the specified string.
- String getLabel()- It fetches the label of the button.
- void addNotify()0 It creates the peer of the button.
- AccessibleContext getAccessibleContext()-It fetched the accessible context associated with the button.
- void addActionListener(ActionListener l)-It adds the specified action listener to get the action events from the button.
- String getActionCommand()-It returns the command name of the action event fired by the button.

- ActionListener[ ] getActionListeners()-It returns an array of all the action listeners registered on the button.
- T[ ] getListeners(Class listenerType)-It returns an array of all the objects currently registered as FooListeners upon this Button
- protected String paramString()-It returns the string which represents the state of button.
- protected void processActionEvent (ActionEvent e)- It process the action events on the button by dispatching them to a registered ActionListener object.
- protected void processEvent (AWTEvent e) - It process the events on the button
- void removeActionListener (ActionListener l)-It removes the specified action listener so that it no longer receives action events from the button.
- void setActionCommand(String command)- It sets the command name for the action event given by the button.

```
import java.awt.*;
public class ButtonExample {
public static void main (String[] args) {

   // create instance of frame with the label
   Frame f = new Frame("Button Example");

   // create instance of button with label
   Button b = new Button("Click Here");

   // set the position for the button in frame
   b.setBounds(50,100,80,30);

   // add button to the frame
   f.add(b);
   // set size, layout and visibility of frame
```

```
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```


## 8. CHECK BOX

The checkbox control turns on (true) or off (false) an option. Each checkbox has a label that describes what the checkbox performs. A checkbox's status can be modified by clicking on it. In this article, we shall be exhaustively learning about AWT CheckBox and several examples to understand it more effortlessly. To make a checkbox, use the Checkbox class. It's used to turn a feature on (true) or off (false). A Checkbox's status changes as you click it, from "on" to "off" or "off" to "on."

**Declaration**

```
public class Checkbox extends Component implements ItemSelectable, Accessible
```

**Checkbox Class Constructors**

- Checkbox()-It constructs a checkbox with no string as the label.
- Checkbox(String label)-It constructs a checkbox with the given label.
- Checkbox(String label, boolean state)-It constructs a checkbox with the given label and sets the given state.
- Checkbox(String label, boolean state, CheckboxGroup group)-It constructs a checkbox with the given label, set the given state in the specified checkbox group.

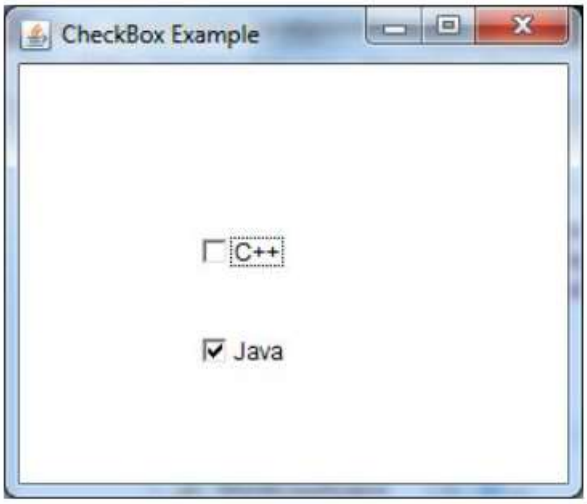
- Checkbox(String label, CheckboxGroup group, boolean state)-It constructs a checkbox with the given label, in the given checkbox group and set to the specified state.

**Checkbox Class Methods**

- void addItemListener(ItemListener IL)- It adds the given item listener to get the item events from the checkbox.
- AccessibleContext getAccessibleContext()-It fetches the accessible context of checkbox.
- void addNotify()- It creates the peer of checkbox.
- CheckboxGroup getCheckboxGroup()- It determines the group of checkbox.
- ItemListener[] getItemListeners()-It returns an array of the item listeners registered on checkbox.
- String getLabel()-It fetched the label of checkbox.
- T[] getListeners(Class listenerType)- It returns an array of all the objects registered as FooListeners.
- Object[] getSelectedObjects()-It returns an array (size 1) containing checkbox label and returns null if checkbox is not selected.
- boolean getState()-It returns true if the checkbox is on, else returns off.
- protected String paramString()- It returns a string representing the state of checkbox.
- protected void processEvent(AWTEvent e)- It processes the event on checkbox.
- protected void processItemEvent(ItemEvent e)-It process the item events occurring in the checkbox by dispatching them to registered ItemListener object.
- void removeItemListener(ItemListener l)-It removes the specified item listener so that the item listener doesn't receive item events from the checkbox anymore.
- void setCheckboxGroup(CheckboxGroup g)- It sets the checkbox's group to the given checkbox.
- void setLabel(String label)-It sets the checkbox's label to the string argument.

- void setState(boolean state)-It sets the state of checkbox to the specified state.

```
import java.awt.*;
public class CheckboxExample1
{
// constructor to initialize
    CheckboxExample1() {
// creating the frame with the title
      Frame f = new Frame("Checkbox Example");
// creating the checkboxes
       Checkbox checkbox1 = new Checkbox("C++");
       checkbox1.setBounds(100, 100,  50, 50);
       Checkbox checkbox2 = new Checkbox("Java", true);
// setting location of checkbox in frame
checkbox2.setBounds(100, 150,  50, 50);
// adding checkboxes to frame
       f.add(checkbox1);
       f.add(checkbox2);

// setting size, layout and visibility of frame
       f.setSize(400,400);
       f.setLayout(null);
       f.setVisible(true);
    }
// main method
public static void main (String args[])
{
   new CheckboxExample1();
}
}
```




## 9. CHECK BOX GROUP CONTROLS

Java AWT CheckboxGroup is a class that is used to create checkboxes and group a set of checkbox buttons together. Many components in the CheckboxGroup separate, which serves as a means to group a set of checkboxes.

```
public class CheckboxGroup extends Object implements Serializable
```

**Constructors:**

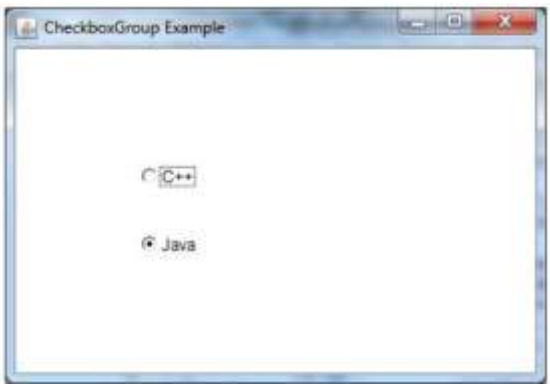
- CheckboxGroup() -This constructor creates a new instance of CheckboxGroup.

**Class Methods in Java AWT CheckboxGroup**

- Checkbox getCurrent()- (Deprecated): As of JDK version 1.1, this method is deprecated and replaced by getSelectedCheckbox()
- Checkbox getSelectedCheckbox()-(Deprecated): As of JDK version 1.1, this method is deprecated and replaced by setSelectedCheckbox(Checkbox)
- void setSelectedCheckbox(Checkbox box)-Sets the currently selected checkbox in this group to be the specified checkbox.
- String toString()- Returns a string representation of this checkbox group, including the value of its current selection.

```
import java.awt.*;
public class CheckboxGroupExample
```

```
{
      CheckboxGroupExample(){
      Frame f= new Frame("CheckboxGroup Example");
       CheckboxGroup cbg = new CheckboxGroup();
       Checkbox checkBox1 = new Checkbox("C++", cbg, false);
       checkBox1.setBounds(100,100, 50,50);
       Checkbox checkBox2 = new Checkbox("Java", cbg, true);
       checkBox2.setBounds(100,150, 50,50);
       f.add(checkBox1);
       f.add(checkBox2);
       f.setSize(400,400);
       f.setLayout(null);
       f.setVisible(true);
    }
public static void main(String args[])
{
   new CheckboxGroupExample();
}
}
```




### 10. CHOICE CONTROL

The choice becomes one of the essential aspects of programming when choosing one of them based on the user's choice. The choice control displays a popup menu of options. The top of the menu shows the selected option. The Java Abstract Window

Toolkit includes the Choice class (AWT). The Choice class displays a popup menu to the user, from which the user can pick an item. The selected item is shown at the top of the list. The Choice class inherits the Component class.

**Declaration**

```
public class Choice extends Component implements ItemSelectable, Accessible
```

**Class constructors**

- Choice()- Creates a new choice menu.

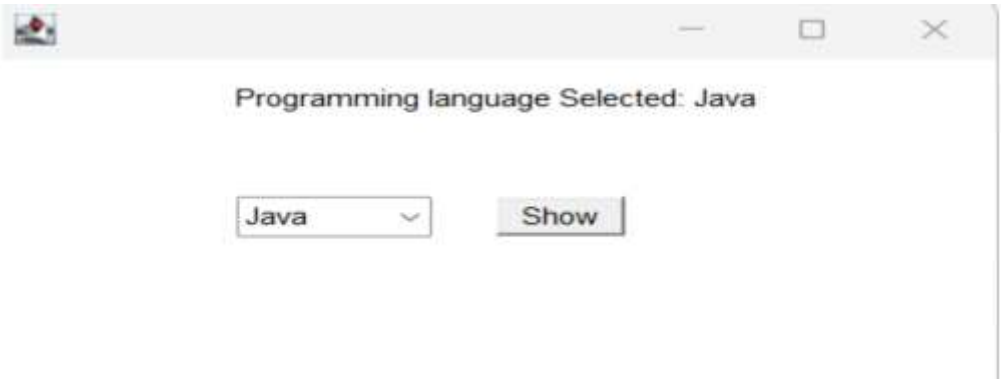
**Class methods**

- void add(String item)- Adds an item to this Choice menu.
- void addItem(String item)- Obsolete as of Java 2 platform v1.1.
- void addItemListener(ItemListener l)- Adds the specified item listener to receive item events from this Choice menu.
- void addNotify()-Creates the Choice's peer.
- int countItems()- Deprecated. As of JDK version 1.1, replaced by getItemCount().
- AccessibleContext getAccessibleContext()-Gets the AccessibleContext associated with this Choice.
- String getItem(int index)-Gets the string at the specified index in this Choice menu.
- int getItemCount()-Returns the number of items in this Choice menu.
- ItemListener[] getItemListeners()-Returns an array of all the item listeners registered on this choice.
- <T extends EventListener> T[] getListeners(Class<T> listenerType)-Returns an array of all the objects currently registered as FooListeners upon this Choice.
- int getSelectedIndex()-Returns the index of the currently selected item.
- String getSelectedItem()-Gets a representation of the current choice as a string.
- Object[] getSelectedObjects()-Returns an array (length 1) containing the currently selected item.

- void insert(String item, int index)-Inserts the item into this choice at the specified position.
- protected String paramString()-Returns a string representing the state of this Choice menu.
- protected void processEvent(AWTEvent e)-Processes events on this choice.
- protected void processItemEvent(ItemEvent e)-Processes item events occurring on this Choice menu by dispatching them to any registered ItemListener objects.
- void remove(int position)-Removes an item from the choice menu at the specified position.
- void remove(String item)-Removes the first occurrence of item from the Choice menu.
- void removeAll()-Removes all items from the choice menu.
- void removeItemListener(ItemListener l)-Removes the specified item listener so that it no longer receives item events from this Choice menu.
- void select(int pos)-Sets the selected item in this Choice menu to be the item at the specified position.
- void select(String str)-Sets the selected item in this Choice menu to be the item whose name is equal to the specified string.

```
import java.awt.*;
import java.awt.event.*;
public class AWTChoiceDemo {
  AWTChoiceDemo() {
      Frame f = new Frame();
    final Label label = new Label();
    label.setAlignment(Label.CENTER);
   label.setSize(400, 100);
    Button b = new Button("Show");
    b.setBounds(200, 100, 50, 20);
    final Choice c = new Choice();
   c.setBounds(100, 100, 75, 75);
```

```
    c.add("C");
    c.add("C++");
    c.add("Java");
    c.add("PHP");
    c.add("Android");
    f.add(c);
    f.add(label);
    f.add(b);

    f.setSize(400, 400);
    f.setLayout(null);
    f.setVisible(true);

    b.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
     String data = "Programming language Selected: "+
c.getItem(c.getSelectedIndex());
     label.setText(data);
    }
    });
    }
 public static void main(String args[])
{
new AWTChoiceDemo();
}
}
```




**11. LIST CONTROL**

The List is a GUI component that displays a list of String values from which the user can select one or more values. The list can be configured as per the choice, and it can be configured as 'single select' or 'multiple select.' It inherits a component class. The AWT list is a component in AWT that extends the Component class. It is used to create a list with multiple values, and from those lists, a user can select any of the values. When the values are selected, then an ItemEvent gets generated, which is handled by implementing the ItemListener interface. Selecting and deselecting items from the list can be done at runtime if any class is implementing this interface.

**Declaration:**

```
public class List extends Component implements ItemSelectable, Accessible
```

**Constructor:**

- List()- Creates a new scrolling list.
- List(int rows)- Creates a new scrolling list initialized with the specified number of visible lines.
- List(int rows, boolean multipleMode)- Creates a new scrolling list initialized to display the specified number of rows.

**Class methods**

- <T extends EventListener> T[] getListeners(Class<T> listenerType)-Returns an array of all the objects currently registered as FooListeners upon this List.
- void add(String item)-Adds the specified item to the end of scrolling list.
- void add(String item, int index)-Adds the specified item to the the scrolling list at the position indicated by the index.
- void addActionListener(ActionListener l)-Adds the specified action listener to receive action events from this list.
- void addItem(String item)-Deprecated. replaced by add(String).
- void addItem(String item, int index)-Deprecated. replaced by add(String, int).
- void addItemListener(ItemListener l)-Adds the specified item listener to receive item events from this list.
- void addNotify()-Creates the peer for the list.
- boolean allowsMultipleSelections()-Deprecated. As of JDK version 1.1, replaced by isMultipleMode().
- void clear()-Deprecated. As of JDK version 1.1, replaced by removeAll().
- int countItems()-Deprecated. As of JDK version 1.1, replaced by getItemCount().
- void delItem(int position)-Deprecated. replaced by remove(String) and remove(int).
- void delItems(int start, int end)- Deprecated. As of JDK version 1.1, Not for public use in the future. This method is expected to be retained only as a package private method.
- void deselect(int index)-Deselects the item at the specified index.
- AccessibleContext getAccessibleContext()-Gets the AccessibleContext associated with this List.
- ActionListener[] getActionListeners()-Returns an array of all the action listeners registered on this list.
- String getItem(int index)-Gets the item associated with the specified index.
- int getItemCount()-Gets the number of items in the list.
- ItemListener[] getItemListeners()-Returns an array of all the item listeners registered on this list.
- String[] getItems()-Gets the items in the list.
- Dimension getMinimumSize()-Determines the minimum size of this scrolling list.

- Dimension getMinimumSize(int rows)-Gets the minumum dimensions for a list with the specified number of rows.
- Dimension getPreferredSize()-Gets the preferred size of this scrolling list.
- Dimension getPreferredSize(int rows)-Gets the preferred dimensions for a list with the specified number of rows.
- int getRows()-Gets the number of visible lines in this list.
- int getSelectedIndex()-Gets the index of the selected item on the list,
- int[] getSelectedIndexes()-Gets the selected indexes on the list.
- String getSelectedItem()-Gets the selected item on this scrolling list.
- String[] getSelectedItems()-Gets the selected items on this scrolling list.
- Object[] getSelectedObjects()-Gets the selected items on this scrolling list in an array of Objects.
- int getVisibleIndex()-Gets the index of the item that was last made visible by the method makeVisible.
- boolean isIndexSelected(int index)-Determines if the specified item in this scrolling list is selected.
- boolean isMultipleMode()- Determines whether this list allows multiple selections.
- void makeVisible(int index)-Makes the item at the specified index visible.
- protected String paramString()-Returns the parameter string representing the state of this scrolling list.
- protected void processActionEvent(ActionEvent e)-Processes action events occurring on this component by dispatching them to any registered ActionListener objects.
- protected void processItemEvent(ItemEvent e)-Processes item events occurring on this list by dispatching them to any registered ItemListener objects.
- void remove(int position)-Removes the item at the specified position from this scrolling list.
- void remove(String item)-Removes the first occurrence of an item from the list.
- void removeActionListener(ActionListener l)-Removes the specified action listener so that it no longer receives action events from this list
- void removeAll()-Removes all items from this list.

- void removeItemListener(ItemListener l)-Removes the specified item listener so that it no longer receives item events from this list.
- void removeNotify()-Removes the peer for this list.
- void replaceItem(String newValue, int index)-Replaces the item at the specified index in the scrolling list with the new string.
- void select(int index)-Selects the item at the specified index in the scrolling list.
- void setMultipleMode(boolean b)- Sets the flag that determines whether this list allows multiple selections.

```
// importing awt class
import java.awt.*;

public class ListExample1 {
  // class constructor
  ListExample1() {
    // creating the frame
    Frame f = new Frame();
    // creating the list of 5 rows
    List l1 = new List(5);

    // setting the position of list component
    l1.setBounds(100, 100, 75, 75);

    // adding list items into the list
    l1.add("Item 1");
    l1.add("Item 2");
    l1.add("Item 3");
    l1.add("Item 4");
    l1.add("Item 5");

    // adding the list to frame
    f.add(l1);
```

```
    // setting size, layout and visibility of frame
    f.setSize(400, 400);
    f.setLayout(null);
    f.setVisible(true);
  }

  // main method
  public static void main(String args[]) {
    new ListExample1();
  }
}
```