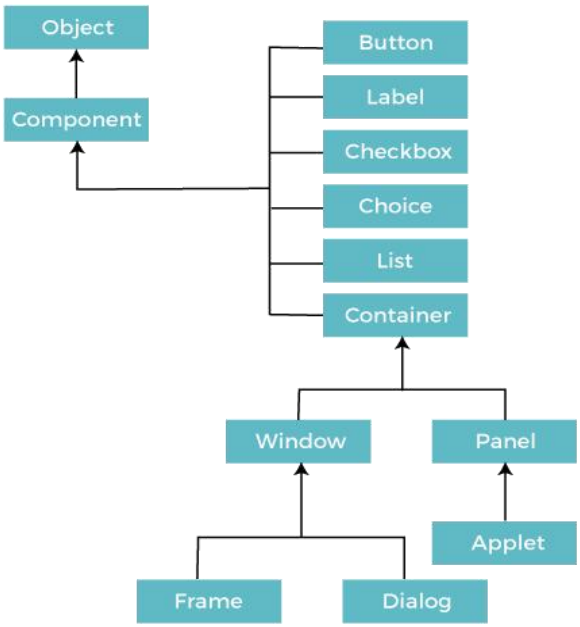
Java AWT

Java AWT (Abstract Window Toolkit) is an API to develop Graphical User Interface (GUI) or windows-based applications in Java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavy weight i.e. its components are using the resources of underlying operating system (OS).

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

Java AWT Hierarchy



Components

All the elements like the button, text fields, scroll bars, etc. are called components. In Java AWT, there are classes for each component as shown in above diagram. In order to place every component in a particular position on a screen, we need to add them to a container.

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**.

It is basically a screen where the where the components are placed at their specific locations. Thus it contains and controls the layout of components.

Types of containers:

There are four types of containers in Java AWT:

1. Window
2. Panel
3. Frame
4. Dialog

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. We need to create an instance of Window class to create this container.

Panel

The Panel is the container that doesn't contain title bar, border or menu bar. It is generic container for holding the components. It can have other components like button, text field etc. An instance of Panel class creates a container, in which we can add components.

Frame

The Frame is the container that contain title bar and border and can have menu bars. It can have other components like button, text field, scrollbar etc. Frame is most widely used container while developing an AWT application.

Useful Methods of Component Class

Method	Description
public void add(Component c)	Inserts a component on this component.
public void setSize(int width,int height)	Sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	Defines the layout manager for the component.
public void setVisible(boolean status)	Changes the visibility of the component, by default false.

1. Label

Syntax

public class Label extends Component implements Accessible

Label Class Constructors

1. Label():

Creates Empty Label.

2. Label(String str):

Constructs a Label with str as its name.

3. Label(String str, int x):

Constructs a label with the specified string and x as the specified alignment

2. Button

AWT Button is a control component with a label that generates an event when clicked on. Button Class is used for creating a labeled button that is platform-independent.

Syntax

public class Button extends Component implements Accessible

Button Class Constructors

1. Button( ):

Creates a Button with no label i.e. showing an empty box as a button.

2. Button(String str):

Creates a Button with String str as a label. For example if str=”Click Here” button with show click here as the value.

3. TextField

Syntax

public class TextField extends TextComponent

TextField Class constructors

1. TextField():

Constructs a TextField component.

2. TextField(String text):

Constructs a new text field initialized with the given string str to be displayed.

3. TextField(int col):

Creates a new text field(empty) with the given number of columns (col).

4. TextField(String str, int columns):

Creates a new text field(with String str in the display) with the given number of columns (col).

4. Checkbox

Syntax

public class Checkbox extends Component implements ItemSelectable, Accessible

Checkbox Class Constructors

1. Checkbox():

Creates a checkbox with no label.

2. Checkbox(String str):

Creates a checkbox with a str label.

3. Checkbox(String str, boolean state, CheckboxGroup group):

Creates a checkbox with the str label, and sets the state in the mentioned group.

5. CheckboxGroup

CheckboxGroup Class is used to group together a set of Checkbox.

Syntax

public class CheckboxGroup extends Object implements Serializable

Note: CheckboxGroup enables the use of radio buttons in AWT.

6. Choice

The object of the Choice class is used to show a popup menu of choices.

Syntax

public class Choice extends Component implements ItemSelectable, Accessible

Choice Class constructor

Choice(): It creates a new choice menu.

7. List

The object of the AWT List class represents a list of text items.

Syntax

public class List extends Component implements ItemSelectable, Accessible

Class Constructors

***1*. List():**

Creates a new list.

2. List(int row):

Creates lists for a given number of rows(row).

3. List(int row, Boolean Mode)

Ceates new list initialized that displays the given number of rows.

8. Canvas

Syntax

public class Canvas extends Component implements Accessible

Canvas Class Constructors

***1.* Canvas():**

Creates new Canvas.

2. Canvas(GraphicConfiguration config):

It creates a new Canvas with the given Graphic configuration.

9. Scrollbar

Syntax

public class Scrollbar extends Component implements Adjustable, Accessible

Scrollbar Class Constructors

1. **Scrollbar():**

It Creates a new vertical Scrollbar in the Application.

2. Scrollbar(int orientation):

Creates a new vertical Scrollbar with the given orientation.

3. Scrollbar(int orientation, int value, int visible, int mini, int maxi):

Creates a new scrollbar with the orientation mentioned with value as the default value and [mini, maxi] as the lower and higher limit.

10. MenuItem & Menu

MenuItem class adds a simple labeled menu item on the menu. The MenuItem class allows you to create individual items that can be added to menus.

Menu is a component used to create a dropdown menu that can contain a list of MenuItem components.

Syntax of MenuItem

public class MenuItem extends MenuComponent implements Accessible

Syntax of Menu

public class Menu extends MenuItem implements MenuContainer, Accessible

11. PopupMenu

Java AWT PopupMenu is a component that is used for dynamically popping up a menu that appears when the user right-clicks or performs any other action on a component.

Syntax

public class PopupMenu extends Menu implements MenuContainer, Accessible

12. Panel

Java AWT Panel is a container class used to hold and organize graphical components in a Java Application.

Syntax

public class Panel extends Container implements Accessible

13. Toolkit

Java AWT Toolkit class provides us with a platform-independent way to access various system resources and functionalities. Subclasses of Toolkit are used to bind various components.

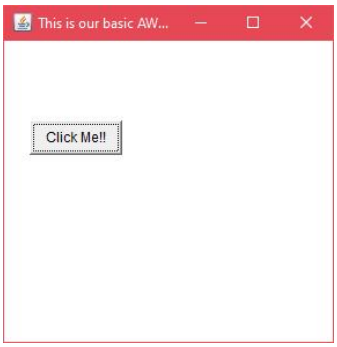
Syntax

public abstract class Toolkit extends Object

AWT Example by Inheritance

```
import java.awt.*;
public class AWTExample1 extends Frame
{
    AWTExample1()
    {
      Button b = new Button("Click Me!!");
      b.setBounds(30,100,80,30);
      add(b);
      setSize(300,300);
      setTitle("This is our basic AWT example");
      setLayout(null);
      setVisible(true);
    }
public static void main(String args[])
{
  AWTExample1 f = new AWTExample1();
 }
}
```

Output:



Event

An **event** can be defined as changing the state of an object or behavior by performing actions. Actions can be a button click, cursor movement, keypress through keyboard or page scrolling, etc.

The **java.awt.event** package can be used to provide various event classes.

Event Class

1. **ActionEvent :** Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
2. **ItemEvent :** Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.
3. **AdjustmentEvent :** Generated when a scroll bar is manipulated.
4. **ComponentEvent :** Generated when a component is hidden, moved, resized, or becomes visible.
5. **ContainerEvent :** Generated when a component is added to or removed from a container.
6. **TextEvent :** Generated when the value of a text area or text field is changed.
7. **FocusEvent :** Generated when a component gains or loses keyboard focus.
8. **KeyEvent :** Generated when input is received from the keyboard.
9. **MouseEvent :** Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
10. **WindowEvent :** Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Event Sources

Events are generated from the source. There are various sources like buttons, checkboxes, list, menu-item, choice, scrollbar, text components, windows, etc., to generate events.

1. **Button :** Generates action events when the button is pressed.
2. **Check box :** Generates item events when the check box is selected or deselected.
3. **List :** Generates action events when an item is double-clicked; generates item events when an item is selected or deselected.
4. **Choice :** Generates item events when the choice is changed.
5. **Scroll bar :** Generates adjustment events when the scroll bar is manipulated.
6. **Text components :** Generates text events when the user enters a character.
7. **Menu item :** Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected.
8. **Window :** Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Event Listener

Listeners are used for handling the events generated from the source. Each of these listeners represents interfaces that are responsible for handling events.

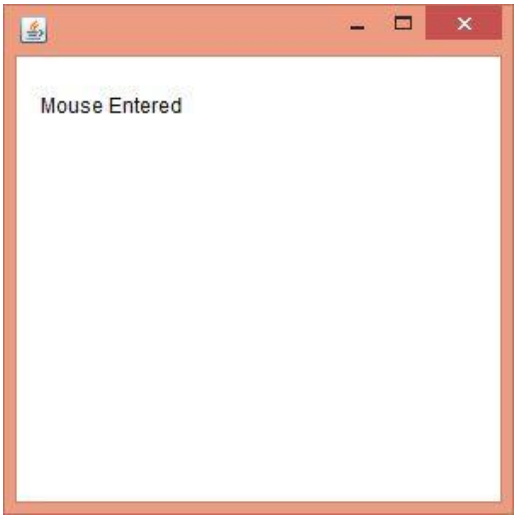
1. **ActionListener :** Defines one method to recognize when action performed.
2. **ItemListener :** Defines one method to recognize when the state of an item changes.
3. **AdjustmentListener :** Defines one method to receive adjustment events.
4. **TextListener :** Defines one method to recognize when a text value changes.
5. **ComponentListener :** Defines four methods to recognize when a component is hidden, moved, resized, or shown.
6. **ContainerListener :** Defines two methods to recognize when a component is added to or removed from a container.
7. **FocusListener :** Defines two methods to recognize when a component gains or loses keyboard focus.
8. **KeyListener :** Defines three methods to recognize when a key is pressed, released, or typed.
9. **MouseListener :** Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.
10. **MouseMotionListener :** Defines two methods to recognize when the mouse is dragged or moved.
11. **MouseWheelListener :** Defines one method to recognize when the mouse wheel is moved.
12. **WindowListener :** Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

MouseListener Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseListenerExample extends Frame implements MouseListener
{
   Label l;
   MouseListenerExample()
  {
      addMouseListener(this);
      l=new Label();
      l.setBounds(20,50,100,20);
      add(l);
      setSize(300,300);
      setLayout(null);
      setVisible(true);
   }
   public void mouseClicked(MouseEvent e)   {      l.setText("Mouse Clicked");     }
   public void mouseEntered(MouseEvent e)   {      l.setText("Mouse Entered");     }
   public void mouseExited(MouseEvent e)    {      l.setText("Mouse Exited");      }
   public void mousePressed(MouseEvent e)   {      l.setText("Mouse Pressed");     }
   public void mouseReleased(MouseEvent e)  {      l.setText("Mouse Released");     }

public static void main(String[] args)
{
 MouseListenerExample ob1 = new MouseListenerExample();
 }
}
```

Output:

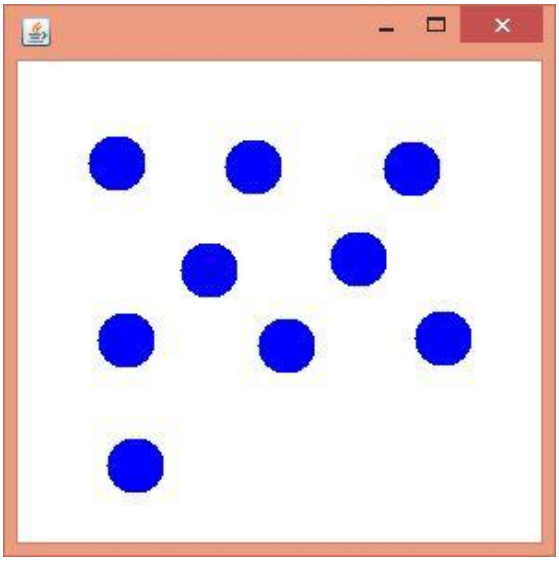


MouseListener Example 2

```
import java.awt.*;
import java.awt.event.*;
public class MouseListenerExample2 extends Frame implements MouseListener
{
   MouseListenerExample2()
   {
      addMouseListener(this);
      setSize(300,300);
      setLayout(null);
      setVisible(true);
   }
   public void mouseClicked(MouseEvent e)
   {
      Graphics g=getGraphics();
      g.setColor(Color.BLUE);
      g.fillOval(e.getX(),e.getY(),30,30);
   }
   public void mouseEntered(MouseEvent e) {}
   public void mouseExited(MouseEvent e) {}
   public void mousePressed(MouseEvent e) {}
   public void mouseReleased(MouseEvent e) {}

public static void main(String[] args)
{
   new MouseListenerExample2();
}
}
```

Output:



ItemListener Example

```
import java.awt.*;
import java.awt.event.*;
public class ItemListenerExample implements ItemListener
{
   Checkbox checkBox1,checkBox2;
   Label label;
   ItemListenerExample()
   {
      Frame f= new Frame("CheckBox Example");
      label = new Label();
      label.setAlignment(Label.CENTER);
      label.setSize(400,100);
      checkBox1 = new Checkbox("C++");
      checkBox1.setBounds(100,100, 50,50);
      checkBox2 = new Checkbox("Java");
      checkBox2.setBounds(100,150, 50,50);
      f.add(checkBox1);
      f.add(checkBox2);
      f.add(label);
      checkBox1.addItemListener(this);
      checkBox2.addItemListener(this);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
    }
   public void itemStateChanged(ItemEvent e)
  {
      if(e.getSource()==checkBox1)
         label.setText("C++ Checkbox: "
         + (e.getStateChange()==1?"checked":"unchecked"));
      if(e.getSource()==checkBox2)
      label.setText("Java Checkbox: "
      + (e.getStateChange()==1?"checked":"unchecked"));
    }
public static void main(String args[])
{
   new ItemListenerExample();
}
}
```

Output: