Course: Advanced Java Programming (22517)

Unit 1

Abstract Windowing Toolkit (AWT)

By

Mr. Vijay M Bande
Lecturer in Computer Engineering
Government Polytechnic Khamgaon

COURSE OUTCOMES(CO'S)

a) **Develop Programs using GUI Framework (AWT and Swings)**

b) Handle events of AWT and Swings Components.

c) Develop programs to handle events in Java Programming

d) Develop Java programs using networking basics.

e) Develop programs using Database.

f) Develop programs using Servlets.

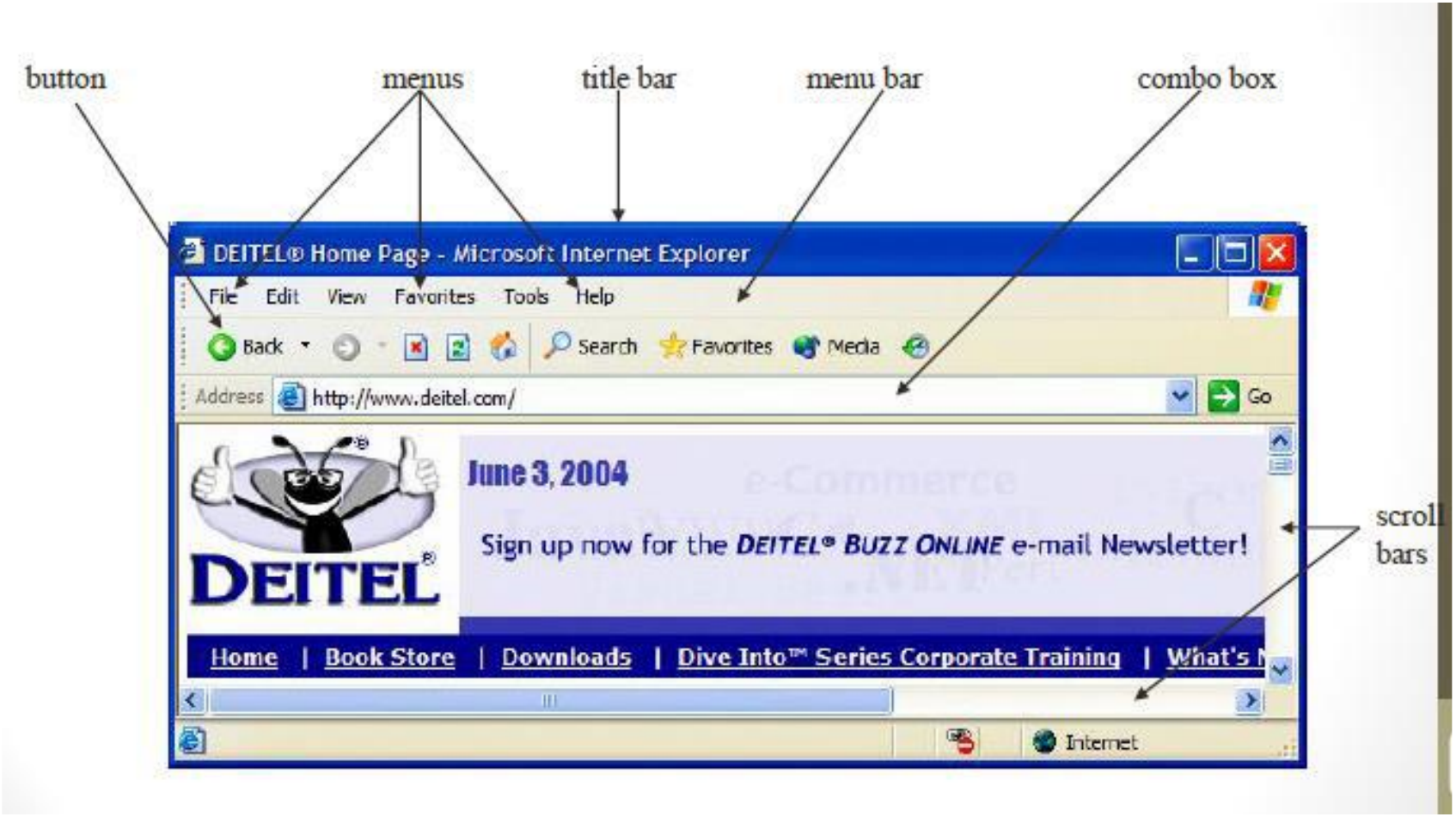
UNIT OUTCOMES(UO'S)

Unit	Unit Outcomes (UOs) (in cognitive domain)	Topics and Sub-topics
Unit – I Abstract Windowing Toolkit(AWT)	1a. Develop Graphical user interface (GUI) programs using AWT components for the given problem. 1b. Create Frame window with the specified AWT components. 1c. Arrange the GUI components using specified layout manager. 1d. Develop a program using menu and Dialog Boxes for the given problem.	1.1 Component, container, window, frame, panel. 1.2 Creating windowed programs and applets. 1.3 AWT controls and layout managers:use of AWT controls: labels, buttons,checkbox, checkbox group, scroll bars, text field, text area. 1.4 Use of layout managers: flowLayout, borderLayout,
		gridLayout, cardLayout, gridbagLayout, menubars, menus, dialog boxes, file dialog.

GUI (GRAPHICAL USER INTERFACE)

- **GUI** offers user interaction via some graphical components.
- Window, Frame, Panel, Button, Textfield, TextArea, Listbox, Combobox, Label, Checkbox etc.
- Using these components we can create an interactive user interface for an application.
- **GUI** provides result to end user in response to raised event.
- **GUI** is entirely based event.

GUI (GRAPHICAL USER INTERFACE)



Object

- **The Object** class is the top most class and parent of all the classes in java by default.

- Every class in java is directly or indirectly derived from the object class.

AWT (ABSTRACT WINDOW TOOLKIT)

- **AWT** contains numerous classes and methods that allow you to create and manage window.

import java.awt.*;

- ***Java AWT*** is *an API to develop **GUI** or **window-based** application 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 uses the resources of system.

AWT CLASSES

Class	Description
AWTEvent	Encapsulates AWT events.
AWTEventMulticaster	Dispatches events to multiple listeners.
BorderLayout	The border layout manager. Border layouts use five components: North, South, East, West, and Center.
Button	Creates a push button control.
Canvas	A blank, semantics-free window.
CardLayout	The card layout manager. Card layouts emulate index cards. Only the one on top is showing.
Checkbox	Creates a check box control.
CheckboxGroup	Creates a group of check box controls.
CheckboxMenuItem	Creates an on/off menu item.
Choice	Creates a pop-up list.
Color	Manages colors in a portable, platform-independent fashion.
Component	An abstract superclass for various AWT components.
Container	A subclass of **Component** that can hold other components.
Cursor	Encapsulates a bitmapped cursor.
Dialog	Creates a dialog window.
Dimension	Specifies the dimensions of an object. The width is stored in **width**, and the height is stored in **height**.
EventQueue	Queues events.
FileDialog	Creates a window from which a file can be selected.

Table 25-1 A Sampling of AWT Classes

AWT CLASSES

Class	Description
FlowLayout	The flow layout manager. Flow layout positions components left to right, top to bottom.
Font	Encapsulates a type font.
FontMetrics	Encapsulates various information related to a font. This information helps you display text in a window.
Frame	Creates a standard window that has a title bar, resize corners, and a menu bar.
Graphics	Encapsulates the graphics context. This context is used by the various output methods to display output in a window.
GraphicsDevice	Describes a graphics device such as a screen or printer.
GraphicsEnvironment	Describes the collection of available **Font** and **GraphicsDevice** objects.
GridBagConstraints	Defines various constraints relating to the **GridBagLayout** class.
GridBagLayout	The grid bag layout manager. Grid bag layout displays components subject to the constraints specified by **GridBagConstraints**.
GridLayout	The grid layout manager. Grid layout displays components in a two-dimensional grid.
Image	Encapsulates graphical images.
Insets	Encapsulates the borders of a container.
Label	Creates a label that displays a string.
List	Creates a list from which the user can choose. Similar to the standard Windows list box.
MediaTracker	Manages media objects.
Menu	Creates a pull-down menu.
MenuBar	Creates a menu bar.
MenuComponent	An abstract class implemented by various menu classes.
MenuItem	Creates a menu item.
MenuShortcut	Encapsulates a keyboard shortcut for a menu item.
Panel	The simplest concrete subclass of **Container**.
Point	Encapsulates a Cartesian coordinate pair, stored in **x** and **y**.
Polygon	Encapsulates a polygon.
PopupMenu	Encapsulates a pop-up menu.
PrintJob	An abstract class that represents a print job.
Rectangle	Encapsulates a rectangle.
Robot	Supports automated testing of AWT-based applications.
Scrollbar	Creates a scroll bar control.
ScrollPane	A container that provides horizontal and/or vertical scroll bars for another component.

Table 25-1 A Sampling of AWT Classes *(continued)*

AWT CLASSES

Class	Description
SystemColor	Contains the colors of GUI widgets such as windows, scroll bars, text, and others.
TextArea	Creates a multiline edit control.
TextComponent	A superclass for **TextArea** and **TextField**.
TextField	Creates a single-line edit control.
Toolkit	Abstract class implemented by the AWT.
Window	Creates a window with no frame, no menu bar, and no title.

Table 25-1 A Sampling of AWT Classes *(continued)*

WINDOW FUNDAMENTALS (AWT CLASS HIERARCHY)

- The **AWT** defines **windows according to a class hierarchy** that adds functionality and specificity 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 application window.

- Figure shows the class hierarchy for **Panel** and **Frame** **(AWT Class Hierarchy)**.

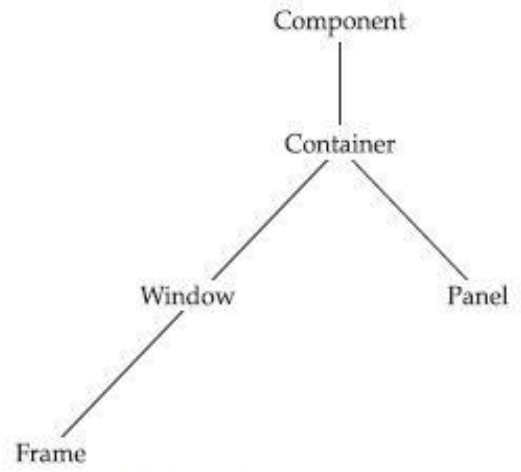


Figure 25-1 The class hierarchy for **Panel** and **Frame**

COMPONENT

- **Component** is an *object having a* ***graphical representation*** that can be displayed on the screen and that can interact with the user.

- **At the top of the AWT hierarchy** is the **Component class.**

- It is **abstract** class that encapsulates all of the attributes of a visual component.

- It ***defines*** over a hundred public methods that are responsible for managing events, such as mouse and keyboard input, positioning and sizing the window, and repainting.

COMPONENT

- **Which object is responsible for remembering the current foreground and background colors and the currently selected text font? (*Answer: Component*)**

- ***It extends:*** Object Class

- ***Implements:*** ImageObserver, MenuContainer, Serializable

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.
void remove(Component c)	Remove a component
void setBounds(int x,int y, int width, int height)	Set the location and size of single component and useful only with null layout.

CONTAINER

- **As the name suggests**, the **Container** object contains the other **AWT** components.
- **Container class is a subclass of Component.**
- It provides additional methods that allow other ***Component to place on it.***
- **Which (Container) is responsible for laying out (that is, positioning) any components that it contains.**
- *It does this through the use of various layout managers*.

CONTAINER

- **Container** is a component in **AWT** that can contain another components like *buttons*, *text fields*, *labels etc*.

- The classes that *extends* **Container** class are known as container such as **Frame**, **Dialog** and **Panel.**

CONTAINERS AND COMPONENTS

- The job of a **Container** is to hold and display Components
- Some common *subclasses* of **Component** are *Button*, *Checkbox*, *Label*, *Scrollbar*, *TextField*, *and TextArea*
- A **Container** is also a **Component**
- Some **Container** subclasses are **Panel** (and Applet, JApplet), **Window** (Frame, JFrame)

CONTAINERS AND COMPONENTS

- The job of a **Container** is to hold and display Components
- Some common *subclasses* of **Component** are *Button*, *Checkbox*, *Label*, *Scrollbar*, *TextField*, *and TextArea*
- A **Container** is also a **Component**
- Some **Container** subclasses are **Panel** (and Applet, JApplet), **Window** (Frame, JFrame)

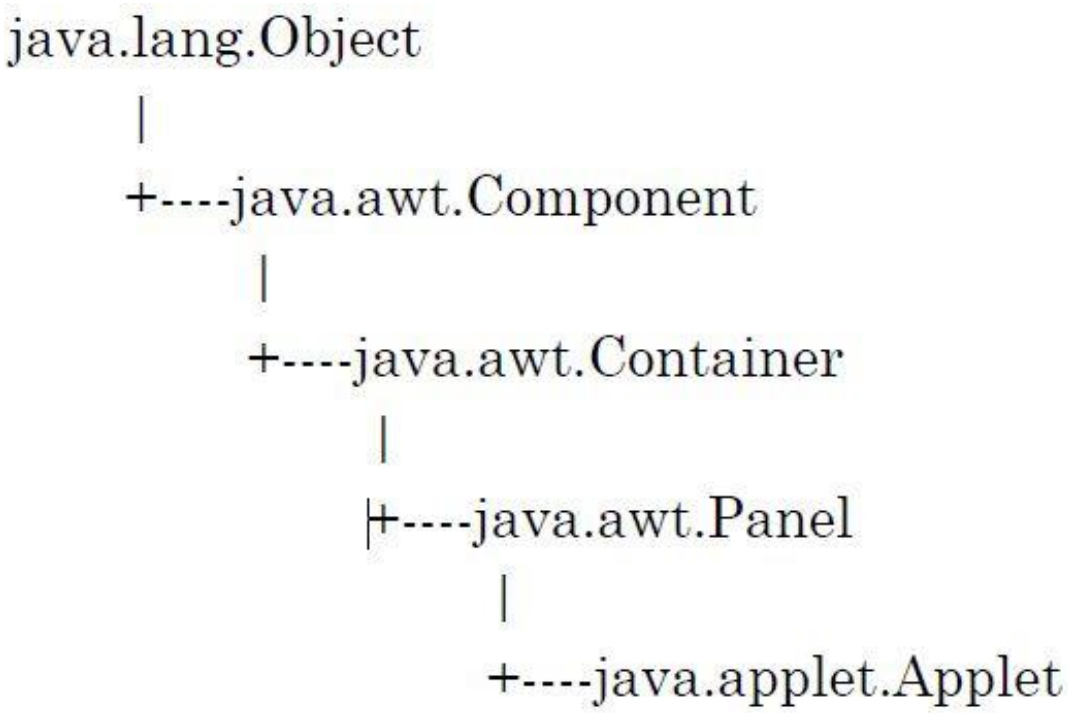
PANEL

- **Panel** is a concrete subclass of **Container.**
- It doesn't add any new methods which simply implements **Container**.
- **Panel** is the *immediate* ***superclass*** of **Applet**.
- When screen output is directed to an applet, it is drawn on the surface of a **Panel** object.

PANEL

- **Panel** is one type of container that ***does not*** contain a *title bar*, *menu bar*, or *border*.
- When you run an applet using an applet viewer, the applet viewer provides the title and border.
- Uses **add()** to add component defined by Container.
- Uses **setSize(), setLocation(), setBounds(), or setPreferredSize()** defined by Component.

AN APPLET IS PANEL IS A CONTAINER



- …so you can display things in an Applet

APPLET

- **Applet** is a *public* class which is predefined by **java.applet.Applet**.
- Here is *no main()* method in **Applet** like Application program. The *main()* method is defined by browser or Appletviewer for Applet.
- Life cycle methods: init, start, paint, stop, destroy
- Applet is one type of container and *subclass* of **Panel. Applet** is *superclass* of **JApplet**.

TO CREATE AN APPLET

- import java.applet.*;
- import java.awt.*;
- Applet code in comment.
- **extends** Applet class
- Life cycle method
- Class must be public

Window

- Creates top-level **Window** means directly on the desktop.
- Don’t create **Window** objects directly.
- The **Window** is the container that have *no borders* and *menu bars*.
- Uses **Frame, Dialog** class which is subclass of **Window** class for creating window.

Frame

- It is *subclass* of **Window** and has a *title bar*, *menu bar*, *borders*, and resizing corners.
- **Frame** object can be created from program or Applet.
- **Through Applet:** Warning message will display *"Java Applet Window"*.
- **Through Program or Application:** Normal window is created.

Working With Frame Windows

- A **Frame** can be created using *two* approaches: They are
  - **Inheritance: By Extending Frame** Class
  - **Association:** Creating an **instance** of **Frame** Class

Working With Frame Windows

- Here are ***TWO*** of **Frame**'s **Constructor** are:
  1. **Frame()**
  2. **Frame(String title)**

- **Notice that** you cannot specify the dimensions of the window. Instead, you must set the size of the window after it has been created.

- A **HeadlessException** is thrown if an attempt is made to create a **Frame** instance in an environment that does not support user interaction.

Working With Frame Windows (Key Methods)

1. Setting and Getting Window Size:

- The **setSize( )** method is used to set the dimensions of the window.
  - void setSize(int width, int height)
  - void **setSize(Dimension** newsize)

Note:-* The dimensions are specified *in terms of **pixels***.

- The **getSize( )** method is used to obtain the current size of a window
  - **Dimension** getSize()

2. 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)

Working With Frame Windows (Key Methods)

3. Setting a Window's Title

- You can change the title in a frame window using **setTitle( ),** which has this general form:

  void setTitle(**String** *newTitle*)

4. Closing a Frame Window:

- When using a frame window, your program must remove that window from the screen when it is closed, by calling **setVisible(false)**.

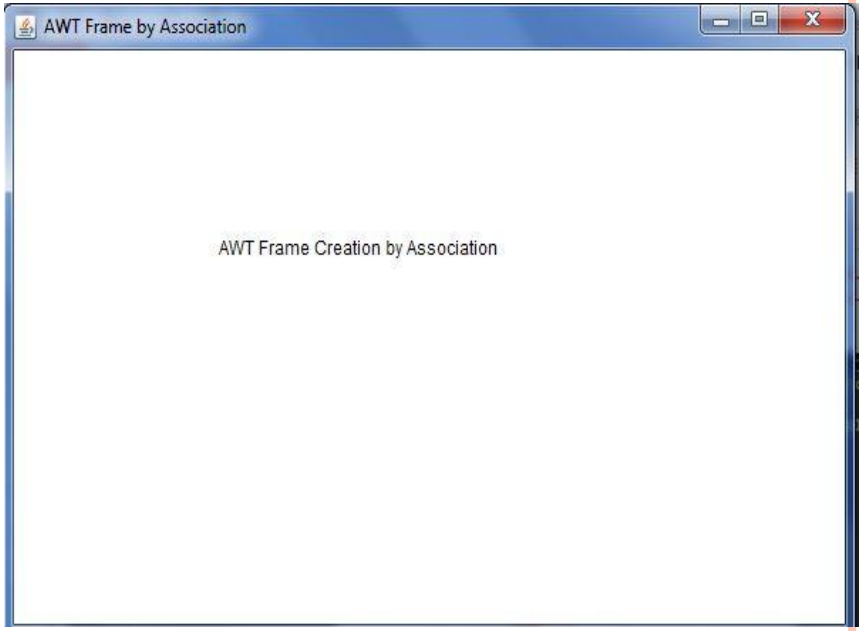
- To intercept a window-close event, you must implement the **windowClosing( )** method of the **WindowListener** interface. Inside **windowClosing( )**, you must remove the window from the screen.

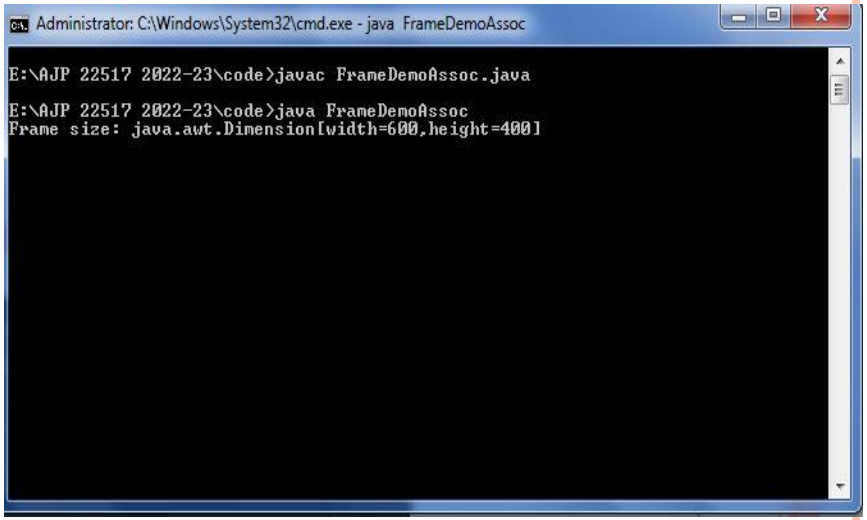
Canvas

- ***Canvas*** class is a part of Java ***AWT***.
- ***Canvas*** is a blank rectangular area where the user can draw or trap input from the user.
- ***Canvas*** class inherits the ***Component*** class.

Creating Windowed Programs and Applets (Example)

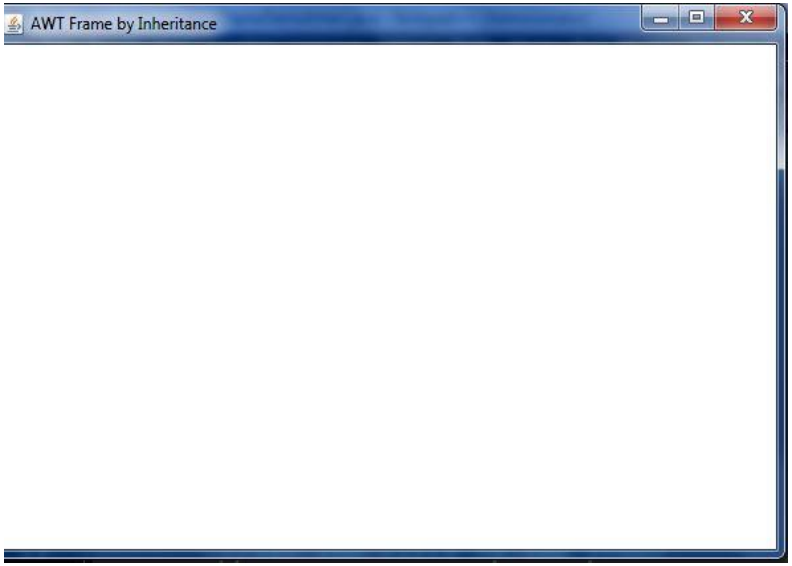
```java
//Demonstrate Frame (By Association)
import java.awt.*;
public class FrameDemoAssoc
{
    public static void main(String main[])
    {
        //Make instance of Frame Class and provides a title
        Frame frame=new Frame("AWT Frame");

        //set width and height of the frame
        frame.setSize(600,400);

        //Show the frame on the screen
        frame.setVisible(true);

        //Get the current size of frame window
        System.out.println("Frame size: "+frame.getSize());

        //Change the title of the frame
        frame.setTitle("AWT Frame by Association");

        //set layout
        frame.setLayout(null);

        //creating a Label
        Label l=new Label("AWT Frame Creation by Association");

        //set the positioning of the Label
        l.setBounds(150,80,250,150);

        //adding components to frame
        frame.add(l);
    }
}
```





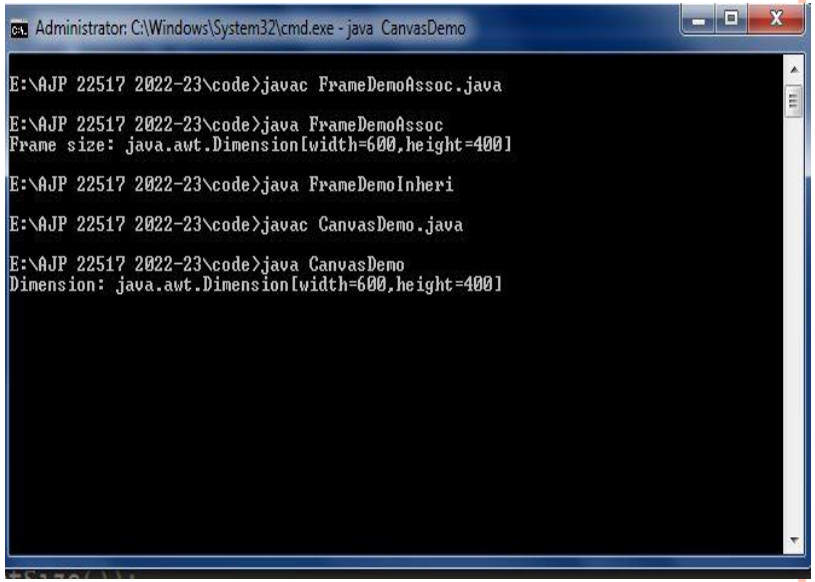
Creating Windowed Programs and Applets (Example)

```java
//Demonstrate Frame (By Inheritance)
import java.awt.*;
public class FrameDemoInheri extends Frame
{
    FrameDemoInheri()
    {
        //set the size frame window
        setSize(600,400);

        //set frame visible on the screen
        setVisible(true);

        //set the title of the frame
         setTitle("AWT Frame by Inheritance");

    }
    public static void main(String args[])
    {
        FrameDemoInheri fdi=new FrameDemoInheri();


    }

}
```



CANVAS (EXAMPLE)

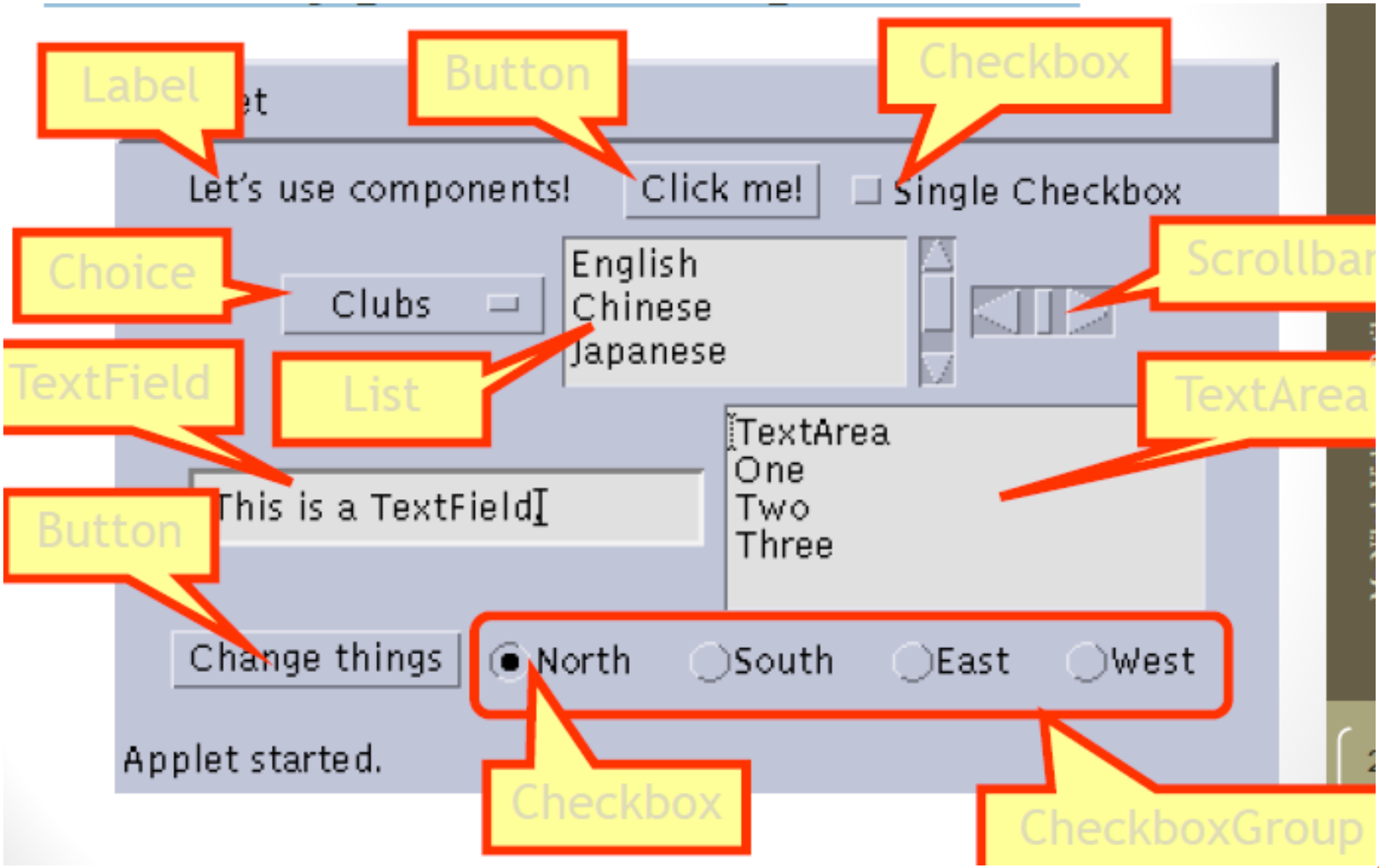
```java
//Creating Canvas
import java.awt.*;
public class CanvasDemo
{
    public CanvasDemo()
    {
        Frame f=new Frame("AWT Canvas");
        f.setSize(600,400);
        f.setVisible(true);
        f.add(new AwtCanvas());
        f.setLayout(null);
        System.out.println("Dimension: "+f.getSize());
    }
    public static void main(String []args)
    {
        new CanvasDemo();
    }
}
class AwtCanvas extends Canvas
{
    AwtCanvas()
    {
        setBackground(Color.orange);
        setSize(400,200);
    }
    public void paint(Graphics g)
    {
        g.setColor(Color.blue);
        g.fillRect(50,70,150,100);
    }
}
```



AWT Controls

- **AWT Controls**: Component which allows you to interact with application.
  - ***Labels***
  - ***Push Buttons***
  - ***Check boxes***
  - ***Choice Lists***
  - ***Lists***
  - ***Scroll bars***
  - ***Text editing***

SOME TYPES OF COMPONENTS



AWT CONTROLS AND LAYOUT MANAGER

- **Layout Manager:** Positioning the components in the container.
  - ***Flow Layout***
  - ***Border Layout***
  - ***Grid Layout***
  - ***Card Layout***
  - ***Grid Bag Layout***

AWT Controls AND Layout Manager

- The appearance of a window is determined by a combination of the controls that it contains and the layout manager used to position them.
- *Menubars, menus, dialog boxes, file dialog.*
- All **AWT controls** are *subclasses* of **Component.**

AWT Controls

- Allows user to interact with application.

- Adding the control in Window
  - First create instance of control.
  - Call add() method defined by container

    Component add(Component compObj)

- Removing the Control
  - Call remove() method defined by container

    void remove(Component obj)
  - For remove all: removeAll() method call.

Responding To Controls

- **Except for labels**, which are passive, all other controls generate events when they are accessed by the user.

- **For example**, when the user clicks on a push button, an event is sent that identifies the push button.

The Headless Exception

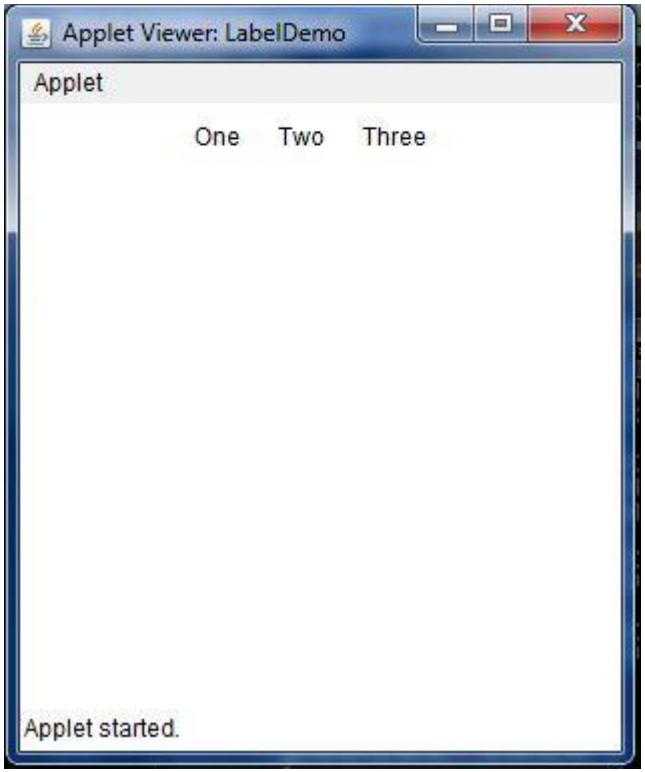
- Most of the **AWT** controls have constructors that can throw a **HeadlessException** when an attempt is made to instantiate a GUI component in a *noninteractive* environment (*such as one in which no display, mouse, or keyboard is present*).

AWT Control: Label

- Used to just display string on window.
- **Passive Component.**
- **Constructors:**
  - Label( )
  - Label(String *str*) ***//left – justified***
  - Label(String *str, int how*) ***// Label.LEFT, Label.RIGHT, Label.CENTER***
- **Methods to perform operation: Setter and Getter Method.**
  - **About text:**
    - void setText(String *str*)
    - String getText( )
  - **About Alignment:**
    - void setAlignment(int *how*)
    - int getAlignment( )

AWT Control: Label (Example)

```java
//Demonstrate Labels
import java.awt.*;
import java.applet.*;
/*
<applet code="LabelDemo" width=300 height=300>
</applet>
*/
public class LabelDemo extends Applet
{
    public void init()
    {
        Label one=new Label("One");
        Label two=new Label("Two");
        Label three=new Label("Three");
        //add labels to applet window
        add(one);
        add(two);
        add(three);
    }
}
```



AWT Control: Button

- *It contains a label and that generates an event when it is pressed.*
- **Active Components**
- **Constructors:**
  - Button( )
  - Button(String *str)*
- **Methods to perform operation: Setter and Getter Method.**
  - void setLabel(String *str)*
  - String getLabel( )

AWT Control: Button (Example)

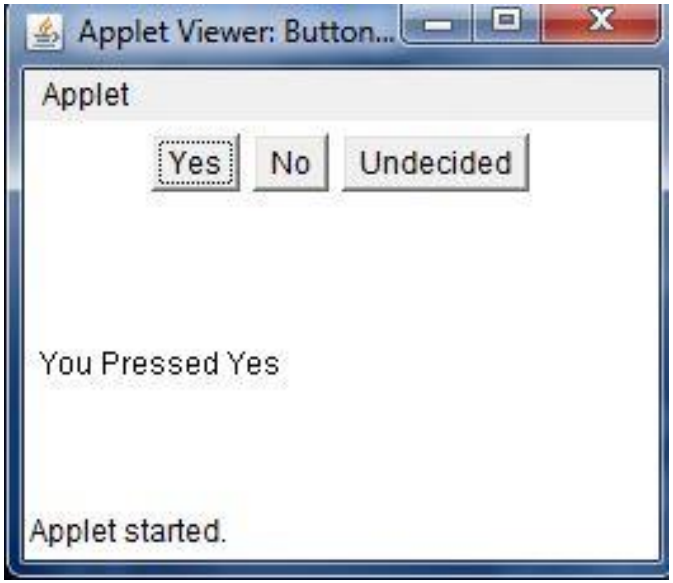
```
//Demonstrate Buttons
import java.awt.*;
import java.applet.*;
/*
<applet code="ButtonDemo" width=250 height=150>
</applet>
*/
public class ButtonDemo extends Applet
{
    public void init()
    {
        Button yes=new Button("Yes");
        Button no=new Button("No");
        Button undecided=new Button("Undecided");

        add(yes);
        add(no);
        add(undecided);
    }
}
```

AWT Control: Handling Buttons

- Each time a button is pressed, an **action event** is generated.
- This is sent to any listeners that previously registered an interest in receiving action event notifications from that component.
- Each listener implements the ***ActionListener*** interface.
- Interface has defined **actionPerformed**() method, called when event is generated.
- **ActionEvent** object is supplied as argument to the method.
- **ActionEvent** object refers both Button and Label of Button
- Label will get by using **getActionCommand() from ActionEvent** which passed.

AWT Control: Handling Buttons (Example)

```
//Demonstrate Buttons with event handling
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*<applet code="ButtonDemo1" width=250 height=150> </applet> */
public class ButtonDemo1 extends Applet implements ActionListener
{
    String msg="";
    Button yes, no, undecided;
    public void init()
    {
        yes=new Button("Yes");
        no=new Button("No");
        undecided=new Button("Undecided");

        add(yes);
        add(no);
        add(undecided);

        yes.addActionListener(this);
        no.addActionListener(this);
        undecided.addActionListener(this);
    }
    public void actionPerformed(ActionEvent ae)
    {
        String str=ae.getActionCommand();
        if(str.equals("Yes"))
            msg="You Pressed Yes";
        else if(str.equals("No"))
            msg="You Pressed No";
        else
            msg="You Pressed Undecided";

        repaint();
    }
    public void paint(Graphics g)
    {
        g.drawString(msg,6,100);
    }
}
```



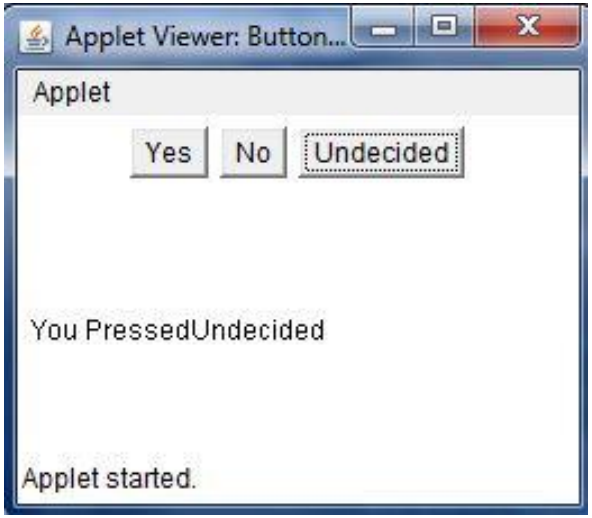
AWT Control: Handling Buttons

- *As mentioned, in addition to comparing button action command strings*, you can also determine which button has been pressed by comparing the object obtained from the **getSource( )** method to the button objects that you added to the window.

- **To do this**, *you must keep a list of the objects when they are added.*

AWT Control: Handling Buttons (Example)

```java
//Recognize Button Objects
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*<applet code="ButtonList" width=250 height=150> </applet> */
public class ButtonList extends Applet implements ActionListener
{
    String msg="";
    Button bList[]=new Button[3];
    public void init()
    {
        Button yes=new Button("Yes");
        Button no=new Button("No");
        Button undecided=new Button("Undecided");

        //store references to buttons as added
        bList[0]=(Button)add(yes);
        bList[1]=(Button)add(no);
        bList[2]=(Button)add(undecided);

        //register to receive action events
        for(int i=0;i<3;i++)
            bList[i].addActionListener(this);
    }
    public void actionPerformed(ActionEvent ae)
    {
        for(int i=0;i<3;i++)
        {
            if(ae.getSource()==bList[i])
                msg="You Pressed"+bList[i].getLabel();
        }
        repaint();
    }
    public void paint(Graphics g)
    {
        g.drawString(msg,6,100);
    }
}
```



AWT Control: Checkbox

- Used to turn an option *on or off.*
- **Small box:** check mark or not.
- Each check box has label.
- You change the state of a check box by clicking on it.
- **Check boxes** can be *used* ***individually*** *or as* ***part of a group.***

AWT Control: Checkbox

- **Constructors are:**
  - **Checkbox( )**
  - **Checkbox(String *str*)**
  - **Checkbox(String *str, boolean on*)**
  - **Checkbox(String *str, boolean on, CheckboxGroup cbGroup*)**
  - **Checkbox(String *str, CheckboxGroup cbGroup, boolean on*)**

AWT Control: Checkbox

- **Setter and Getter methods:**
  - boolean getState( )
  - void setState(**boolean *on*)**
  - String getLabel( )
  - void setLabel(**String *str*)**

- Here, the value of ***on*** determine initial state (true/false).

AWT Control: Checkbox (Example)

```
//Demonstrate check boxes
import java.awt.*;
import java.applet.*;
/*
<applet code="CheckboxDemo" width=240 height=200>
</applet>
*/
public class CheckboxDemo extends Applet
{
    Checkbox windows, solaris, android, mac;
    public void init()
    {
        windows=new Checkbox("Windows",null, true);
        solaris=new Checkbox("Solaris");
        android=new Checkbox("Android",true);
        mac=new Checkbox("Mac");

        add(windows);
        add(solaris);
        add(android);
        add(mac);

    }
}
```

AWT Control: Handling Check Boxes

- Check box is *selected* ***or*** *deselected*, an **item event** is generated.
- This is sent to any listeners that previously registered an interest in receiving item event notifications from that component.
- **For handling** *implements* ***ItemListener*** interface
- ***ItemListener*** interface is defines **itemStateChanged( )** method.
- **ItemEvent** object is supplied as the argument to this method.
- **getState():** Get Status about checkbox.

AWT Control: Handling Check Boxes (Example)

```java
//Demonstrate check boxes with event handling
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/* <applet code="CheckboxDemo1" width=240 height=200></applet> */
public class CheckboxDemo1 extends Applet implements ItemListener
{
    String msg=" ";
    Checkbox windows, solaris, android, mac;
    public void init()
    {
        windows=new Checkbox("Windows",null, true);
        solaris=new Checkbox("Solaris");
        android=new Checkbox("Android",true);
        mac=new Checkbox("Mac");

        add(windows);
        add(solaris);
        add(android);
        add(mac);

        windows.addItemListener(this);
        solaris.addItemListener(this);
        android.addItemListener(this);
        mac.addItemListener(this);
    }
    public void itemStateChanged(ItemEvent ie)
    {
        repaint();
    }

    //Display current state of the check boxes
    public void paint(Graphics g)
    {
        msg="Current State: ";
        g.drawString(msg,6,80);
        msg="Windows: "+windows.getState();
        g.drawString(msg,6,100);
        msg="Solaris: "+solaris.getState();
        g.drawString(msg,6,120);
        msg="Android: "+android.getState();
        g.drawString(msg,6,140);
        msg="Mac: "+mac.getState();
        g.drawString(msg,6,160);

    }
}
```

CheckboxGroup

- It is possible to create a set of ***mutually exclusive*** check boxes in which one and only one check box in the group can be checked at any one time.
- These check boxes are called as ***radio buttons***,
- To create a set of mutually exclusive check boxes, you must first define the group to which they will belong and then specify that group when you construct the check boxes.
- Check box groups are objects of type ***CheckboxGroup***.
- **Only default constructor is defined**, which creates an empty group.

CheckboxGroup

- **Setter and Getter methods:**
  - Checkbox getSelectedCheckbox( )
  - void setSelectedCheckbox(**Checkbox *which***)

Here, *which* is the check box that you want to be selected.

CHECKBOXGROUP (EXAMPLE)

```java
//Demonstrate check box group.
import java.awt.*;
import java.applet.*;
/*
<applet code="CBGroup" width=240 height=200>
</applet>
*/
public class CBGroup extends Applet
{
    Checkbox windows, android, solaris, mac;
    CheckboxGroup cbg;
    public void init()
    {
        cbg=new CheckboxGroup();
        windows=new Checkbox("Windows",cbg, true);
        android=new Checkbox("Android",cbg,false);
        solaris=new Checkbox("Solaris",cbg,false);
        mac=new Checkbox("Mac OS",cbg, false);

        add(windows);
        add(android);
        add(solaris);
        add(mac);
    }

}
```

CheckboxGroup with Event Handling (Example)

```java
//Demonstrate check box group with event handling.
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet code="CBGroup1" width=240 height=200>
</applet>
*/
public class CBGroup1 extends Applet implements ItemListener
{
    String msg="";
    Checkbox windows, android, solaris, mac;
    CheckboxGroup cbg;
    public void init()
    {
        cbg=new CheckboxGroup();
        windows=new Checkbox("Windows",cbg, true);
        android=new Checkbox("Android",cbg,false);
        solaris=new Checkbox("Solaris",cbg,false);
        mac=new Checkbox("Mac OS",cbg, false);

        add(windows);
        add(android);
        add(solaris);
        add(mac);

        windows.addItemListener(this);
        solaris.addItemListener(this);
        android.addItemListener(this);
        mac.addItemListener(this);
    }
    public void itemStateChanged(ItemEvent ie)
    {
        repaint();
    }
    //Display current state of the check boxes.
    public void paint(Graphics g)
    {
        msg="Current Selection: ";
        msg+=cbg.getSelectedCheckbox().getLabel();
        g.drawString(msg,6,100);

    }
}
```