**Advanced Java Programming(22517)**

**Unit 1**

**Introduction to Abstract Windowing Toolkit (AWT) (12 Marks)**

**Unit Outcomes**

* **Develop GUI using awt**
* **Create frame window using different awt component**
* **Arrange GUI components using Layout Manager**
* **Develop program using menu and Dialog Boxes**

**AWT (Abstract Window Toolkit)**

* It is a platform dependent API for creating **Graphical User Interface (GUI)** for java
* AWT contains numerous classes and methods that allow you to create and manage **window**

• import **java.awt.\***;

• 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.

**Abstract Window Toolkit**

A screenshot of a cell phone

Description automatically generated

GUI using Awt

A screenshot of a cell phone

Description automatically generated

**AWT Class Hierarchy**



Applet

Frame

**Simple Applet program**

import java.awt.\*;

import java.applet.\*;

/\*

<applet code="Simpleapplet" width=500 height=200>

</applet>

\*/

public class Simpleapplet extends Applet

{

public void init()

{

setBackground(Color.YELLOW);

}

public void paint(Graphics g)

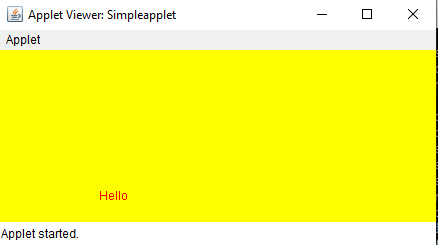
{

g.setColor(Color.RED);

g.drawString("Hello",100,150);

}

}



**Working with Frame Window**

**Frame’s constructors:**

Frame( )

Frame(String *title)*

**Setting the Frame’s Size**

void setSize(int *newWidth, int newHeight)*

void setSize(Dimension *newSize)*

Dimension getSize( )

**Hiding and Showing a Window**

void setVisible(boolean *visibleFlag)*

**Setting a Window’s Title**

void setTitle(String *newTitle)*

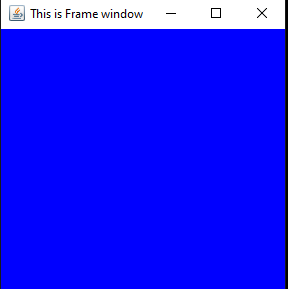
**Closing a Frame Window**

setVisible(false)

To perform window-close event we must implement **windowsClosing()** method of **WindowListener**

//Creating Frame window by extending Frame class

import java.awt.\*;

class Simpleframe extends Frame

{

Simpleframe()

{

setSize(300,300);

setVisible(true);

setTitle("This is Frame window");

setBackground(Color.BLUE);

}

public static void main(String args[])

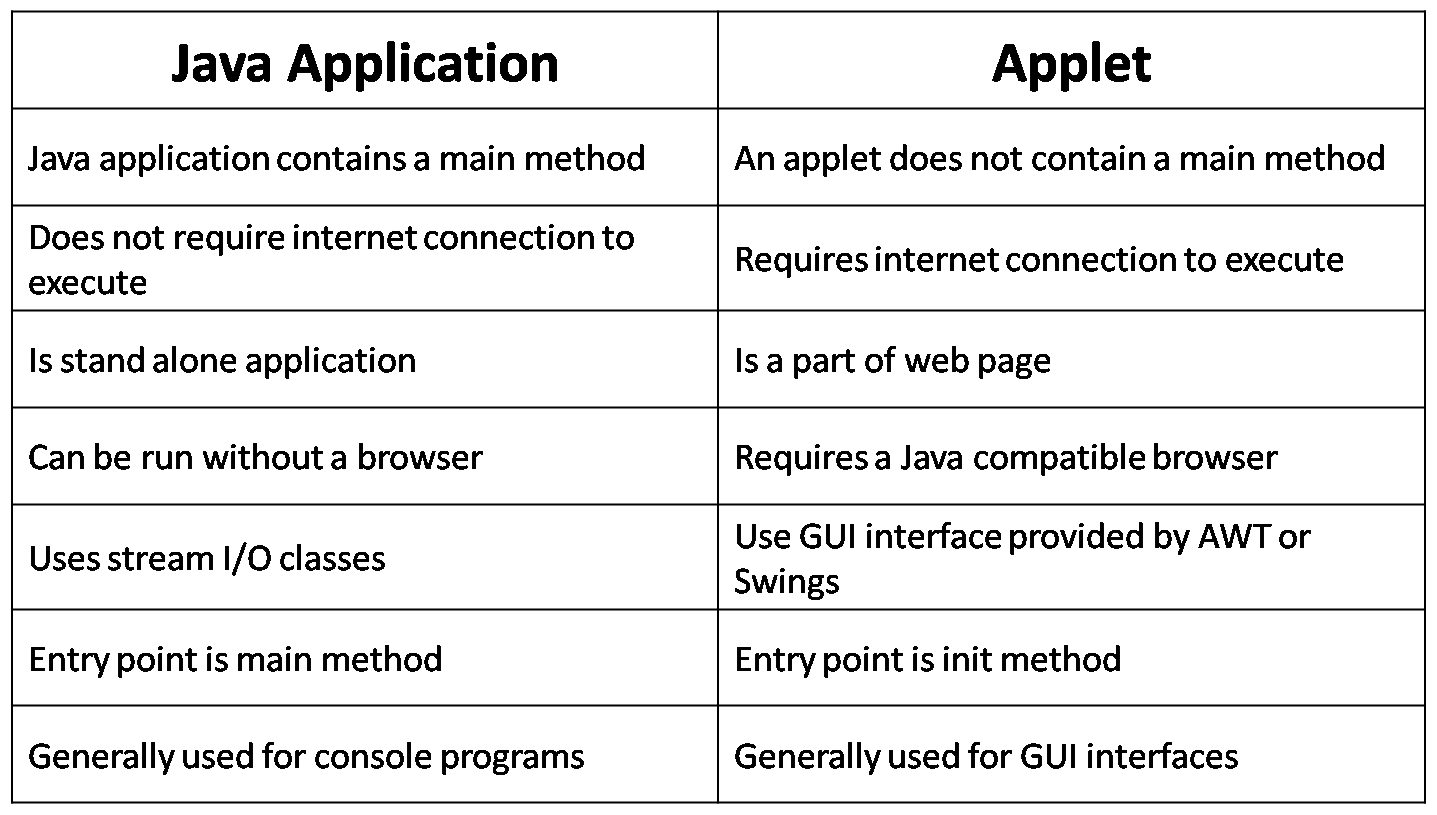
{

Simpleframe f=new Simpleframe();

}

}

Difference between java application(Frame) and Applet



**AWT Class Hierarchy**



**Component**

• Component is an **object** having a graphical representation that **can be displayed on the screen** and that can interact with the user for eg **button,label**

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

* Whichobject 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

**Container**

• **Container** class is a subclass of **Component**.

• Provided additional methods that allow other

**Component** to place on it.

• **Container** is responsible **for laying out (that is, positioning) any components** that it contains.

• It does this through the use of **various layout managers.**

**Panel**

* **Panel** is a concrete subclass of **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.

**Window**

• Creates top-level window means **directly on the desktop**.

• 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.**

**AWT Controls**

• AWT Controls: Component which allows you to interact with application.

• Labels

• Button

• Checkbox

• Checkbox group

• Scrollbars

• Text field

• Text Area

**AWT Control: Label**

• Used to just **display string** on window.

• **Passive Components**

• Constructors:

* Label( )
* Label(String *str*) //left - justified

• Label(String *str*, int *how*) // how value can be Label.LEFT, Label.RIGHT, Label.CENTER

• Methods to perform operation: Setter and Getter Method.

About text:

* void setText(String *str*)
* String getText( )

• About Alighment

* void setAlignment(int *how*)
* int getAlignment( )

**Label(using frame window)**

import **java.awt.\*;**

class Labelframe **extends Frame**

{

**Labelframe()**

{

Label l1=new Label("Name",Label.LEFT);

l1.**setBounds**(30,30,80,30);

**add(l1);**

**setSize**(300,300);

**setTitle**(“frame window”);

**setVisible**(true);

}

**public static void main**(String args[])

{

Labelframe f=new Labelframe();

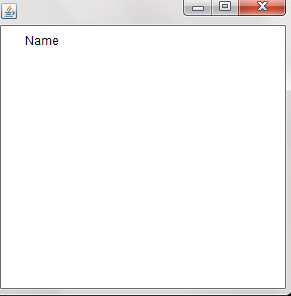
}

}

Run this program in command prompt-

javac Labelframe.java

java Labelframe



**Label(using Applet)**

import java.awt.\*;

import java.applet.\*;

/\*

<applet code="Labelapplet" width=500 height=200>

</applet>\*/

public class Labelapplet extends Applet

{

public void init( )

{

setFont(new Font ( "Times new roman" , Font.BOLD, 14));

Label l1 = new Label("Name", Label.LEFT);

add(l1);

}

}

Output:

**Label(using Applet)**

import java.awt.\*;

import java.applet.\*;

/\*

<applet code="Labelapplet" width=500 height=200>

</applet>\*/

public class Labelapplet extends Applet

{

public void init( )

{

setFont(new Font ( "Times new roman" , Font.BOLD, 14));

Label l1 = new Label("aligned left", Label.LEFT);

add(l1);

Label l2 = new Label("aligned center", Label.CENTER);

add(l2);

Label l3 = new Label("aligned right",Label.RIGHT);

add(l3);

//using setText and getText

Label l4=new Label();

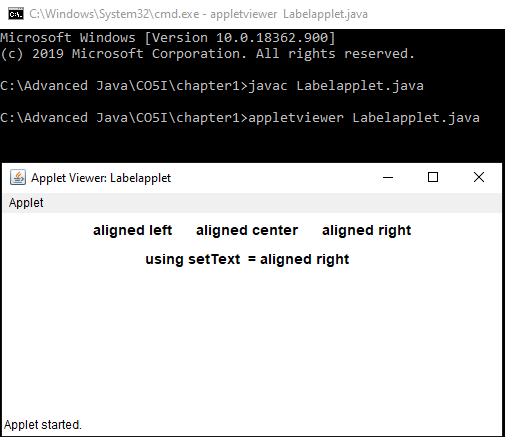
String str=l3.getText();

l4.setText("using setText = "+str);

add(l4);

}

}

****

**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( )

import java.awt.\*;

import java.applet.\*;

/\*

<applet code="Buttonapplet" width=250 height=150>

</applet>

\*/

public class Buttonapplet extends Applet

{

Button b1,b2;

public void init()

{

b1 = new Button("Yes");

b2 = new Button("No");

add(b1);

add(b2);

//getLabel and setLabel

Button b3=new Button();

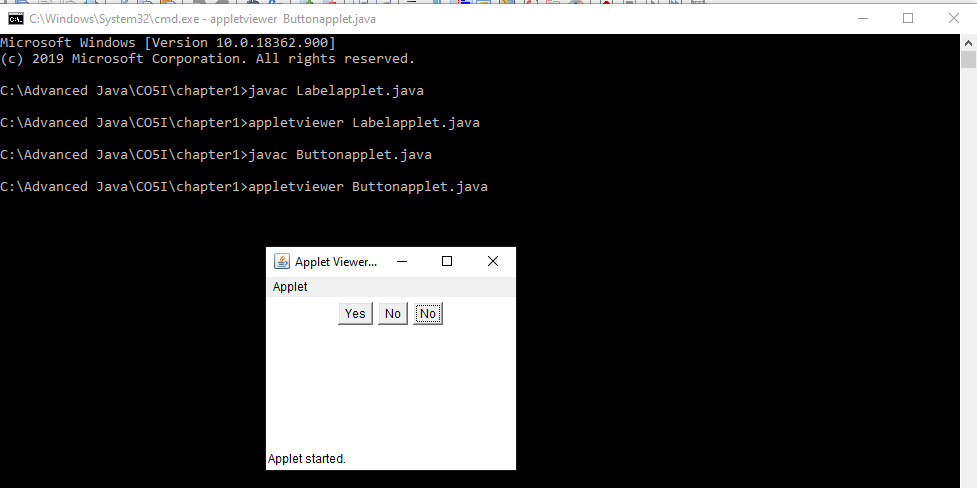
String str=b2.getLabel();

b3.setLabel(str);

add(b3);

}

}



**AWT Control: CheckBox**

•Used to **turn an option on or off**.

•Small box: check mark or not.

•Each check box **has label**.

•Constructors are:

Checkbox( )

Checkbox(String *str*)

Checkbox(String *str*, Boolean *on*)

Checkbox(String *str*, Boolean *on*, CheckboxGroup*name*)

Checkbox(String *str*, CheckboxGroup*name*, Boolean *on*)

* Setter and Getter methods:

Boolean getState( )

void setState(Boolean *on*)

String getLabel( )

void setLabel(String *str*)

**Program**

// Demonstrate check boxes.

import java.awt.\*;

import java.applet.\*;

/\*

<applet code="Checkboxapp" width=250 height=200>

</applet>

\*/

public class Checkboxapp extends Applet

{

Checkbox c1,c2;

public void init()

{

c1 = new Checkbox("Cricket", true);

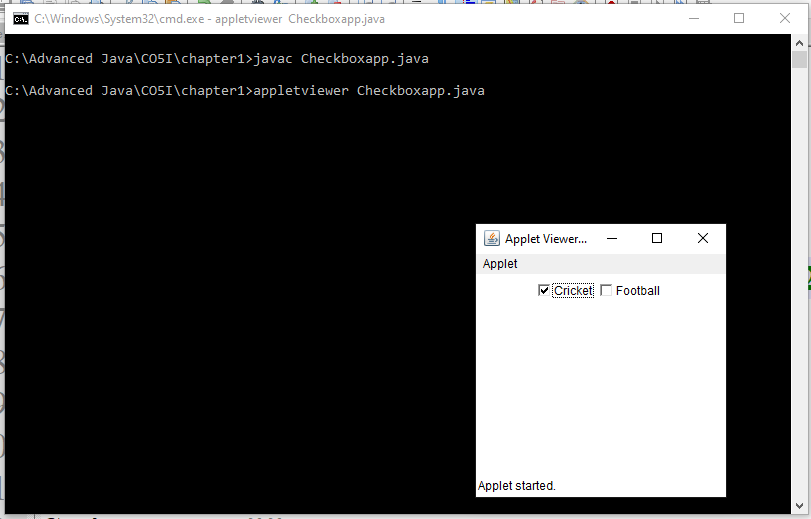
c2= new Checkbox("Football");

add(c1);

add(c2);

}

}



**CheckboxGroup(Radio Button)**

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 often called **radio buttons,**

* Methods

getSelectedCheckbox(),

setSelectedCheckbox().

import java.awt.\*;

import java.applet.\*;

/\*

<applet code="CBGroupapp" width=250 height=200>

</applet>

\*/

public class CBGroupapp extends Applet

{

Checkbox c1,c2 ;

CheckboxGroup cbg;

public void init()

{

cbg = new CheckboxGroup();

c1 = new Checkbox("Male", cbg, true);

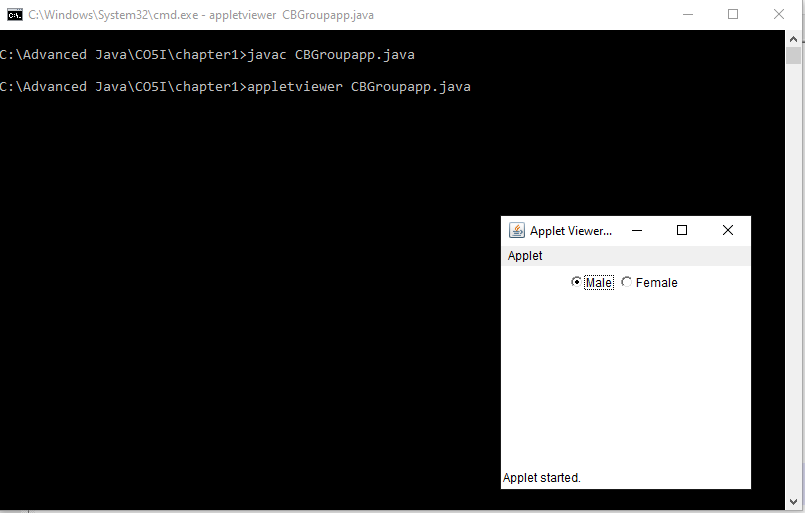
c2= new Checkbox("Female", cbg, false);

add(c1);

add(c2);

}

}

****

**AWT Control: Choice**

•Used to create a **pop-up list** items.

•Default constructor **Choice()** create empty list.

•For add item in list and select active item:

void add(String name)

void select(int *index*)

void select(String *name*)

•Each item in the list is a string that appears as a left-justified label in the order it is added to the **Choice** object.

•To determine selected item:

String getSelectedItem( )

int getSelectedIndex( )

String getItem(int *index*)

* program

import java.awt.\*;

import java.applet.\*;

/\*

<applet code="Choiceapp" width=300 height=180>

</applet>

\*/

public class Choiceapp extends Applet

{

Choice sub;

public void init()

{

sub= new Choice();

// add items to os list

sub.add("c");

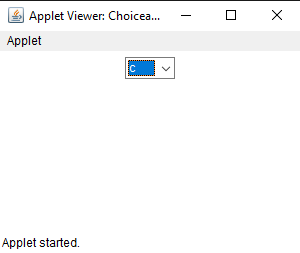
sub.add("c++");

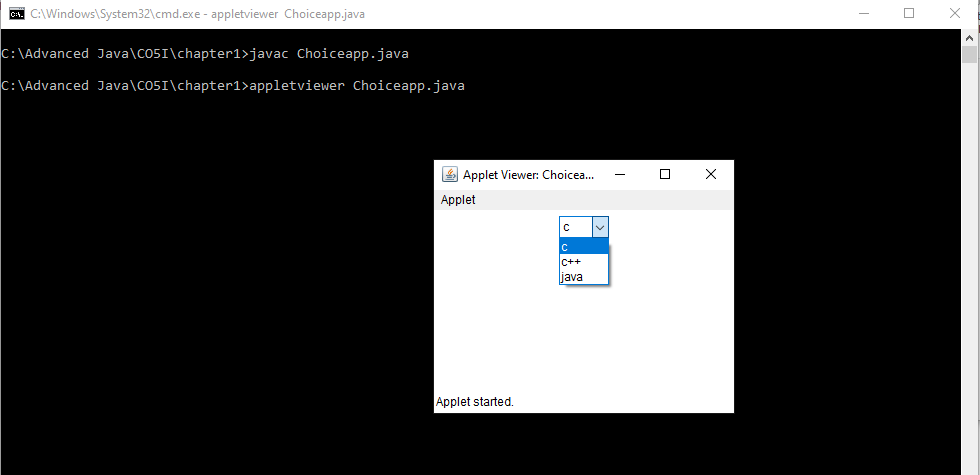
sub.add("java");

add(sub);

}

}



****

**AWT Control: List**

•**List** class provides a compact**, multiple-choice,** scrolling selection list.

•**List** object can be constructed **to show** **any number of choices** in the visible window.

•**In Choice only one item is shown**.

•Constructors

List( )

List(int*numRows*)

List(int*numRows*, boolean*multipleSelect*)

for eg.

**List(4,true)**

* Following methods are used to add items:

void **add**(String *name*)

void **add**(String *name*, int *index*)

•For single selection items:

String **getSelectedItem**( )

Int **getSelectedIndex**( )

•For Multi selection items:

String[ ] **getSelectedItems**( )

int[ ] **getSelectedIndexes**( )

* To retrieve item:

String **getItem**(int *index*)

•To get Item Count

Int **getItemCount**( )

•Active Item

void **select**(int *index*)

**Program-**

import java.awt.\*;

import java.applet.\*;

/\*

<applet code="Listapp" width=300 height=180>

</applet>

\*/

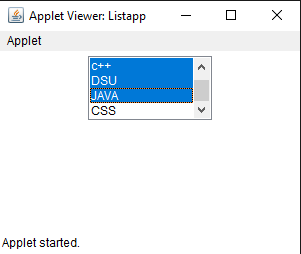
public class Listapp extends Applet

{

List sub;

public void init()

{



// add items to os list

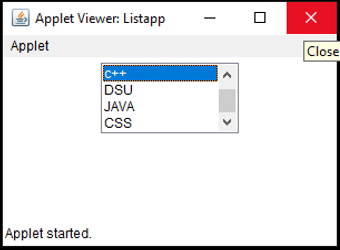
sub.add("c");

sub.add("c++");

sub.add("DSU");

sub.add("JAVA");

sub.add("CSS");



sub.select(1);

// add lists to window

add(sub);

}

}

mport java.awt.\*;

import java.applet.\*;

/\*

<applet code="Listapp" width=300 height=180>

</applet>

\*/

public class Listapp extends Applet

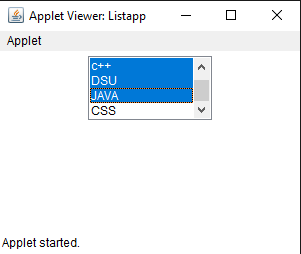
{

List sub;

True means multiple selection is enabled

public void init()

{

sub = new List(4, true);

// add items to os list

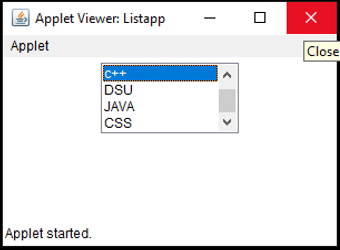
sub.add("c");

sub.add("c++");

sub.add("DSU");

sub.add("JAVA");

sub.add("CSS");



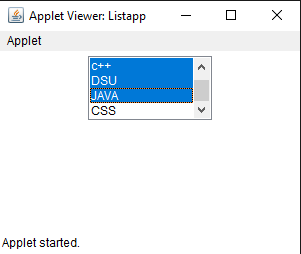
sub.select(1);

// add lists to window

add(sub);

}

}



|  |  |
| --- | --- |
| Choice | List |
|  |  |
|  |  |
| Choice Class is used | List Class is used |
|  |  |

**AWT Control: TextField**

•TextField is subclass of **TextComponent. TextComponent is subclass of Component.**

•**TextField** class implements a **single-line text-entry** area, usually called an ***edit control****.*

•Constructors:

TextField( )

TextField(int *numChars*) -> TextField(**10**)

TextField(String *str*) -> TextField(“enter name”)

TextField(String *str***, int *numChars***) -> TextField(“enter name”**, 10**)

* Setter and Getter Method of TextField and TextComponent:

String **getText( )**

void **setText**(**String *str*)**

* Particular Text selection:

String **getSelectedText( )**

void **select**(int*startIndex*, int*endIndex*)

•About Modification of Text:

boolean **isEditable( )**

void **setEditable**(Boolean *canEdit*)

* Setting echo character to text field and related methods:

void **setEchoChar(char *ch*)**

Boolean **echoCharIsSet**( )

char **getEchoChar**( )

**Program-**

// Demonstrate text field.

import java.awt.\*;

import java.applet.\*;

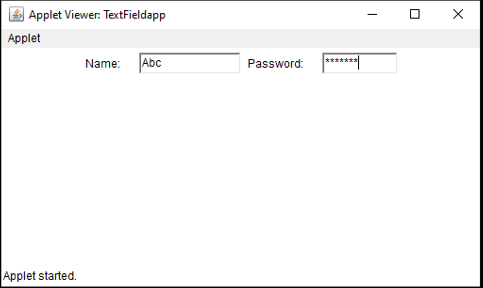
/\*

<applet code="TextFieldapp" width=380 height=150>

</applet>

\*/

public class TextFieldapp extends Applet

{

TextField t1, t2;

Label l1,l2;

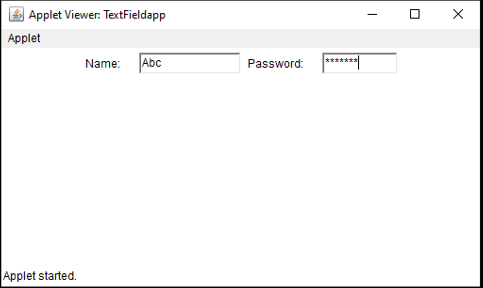
public void init()

{

l1 = new Label("Name: ");

l2 = new Label("Password: ");

t1 = new TextField(12);

t2= new TextField(8);

t2.setEchoChar('\*');

add(l1);

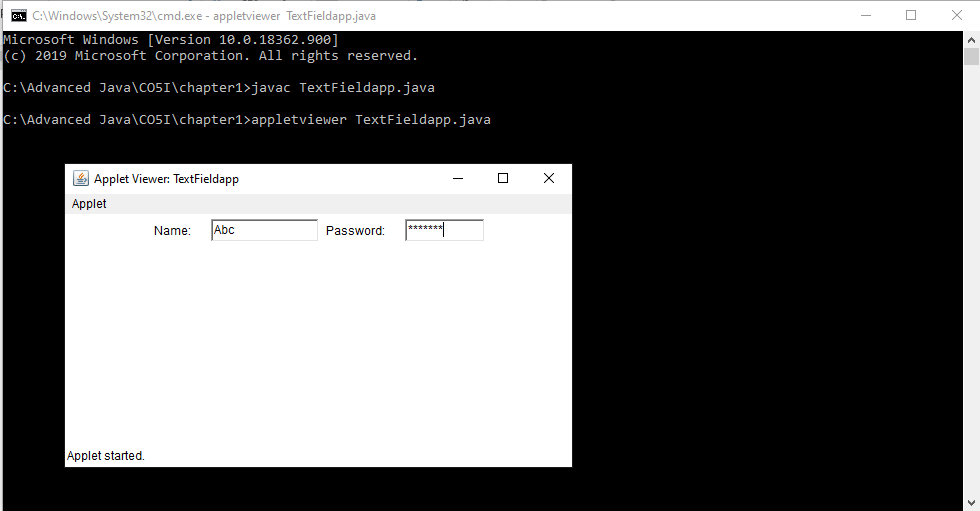
add(t1);

add(l2);

add(t2);

}

}



**AWT Control: TextArea**

•Sometimes a **single line of text input is not enough** for a given task.

•Subclass of **TextComponent**.

•Constructors:

* TextArea( )
* TextArea(int*numLines,* int*numChars*) **-> TextArea(10, 5)**
* TextArea(String *str*) **-> TextArea(“welcome”)**
* TextArea(String *str*, int*numLines*, int*numChars*) -> **TextArea(“welcome”,10, 5)**
* TextArea(String *str*, int*numLines*, int*numChars*, int*sBars*) ->

**TextArea(“welcome”,10, 5, SCROLLBARS\_BOTH)**

* The values of **sbar**:

SCROLLBARS\_BOTH

SCROLLBARS\_NONE

SCROLLBARS\_HORIZONTAL\_ONLY

SCROLLBARS\_VERTICAL\_ONLY

•It supports: **getText( )**, **setText( )**,

**getSelectedText( )**, **select( )**, **isEditable( )**,

and **setEditable( )**

•Other some methods:

void append()

void insert()

void replaceRange()

Program:-

// Demonstrate TextArea.

**import java.awt.\*;**

**import java.applet.\*;**

/\*

<applet code="**TextAreaapp**" width=300 height=250>

</applet>

\*/

public class **TextAreaapp** extends Applet

{

public void **init()**

{

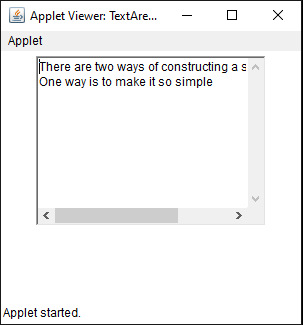
String val = "There are two ways of constructing " +

"a software design.\n" +

"One way is to make it so simple\n" ;

No. of characters

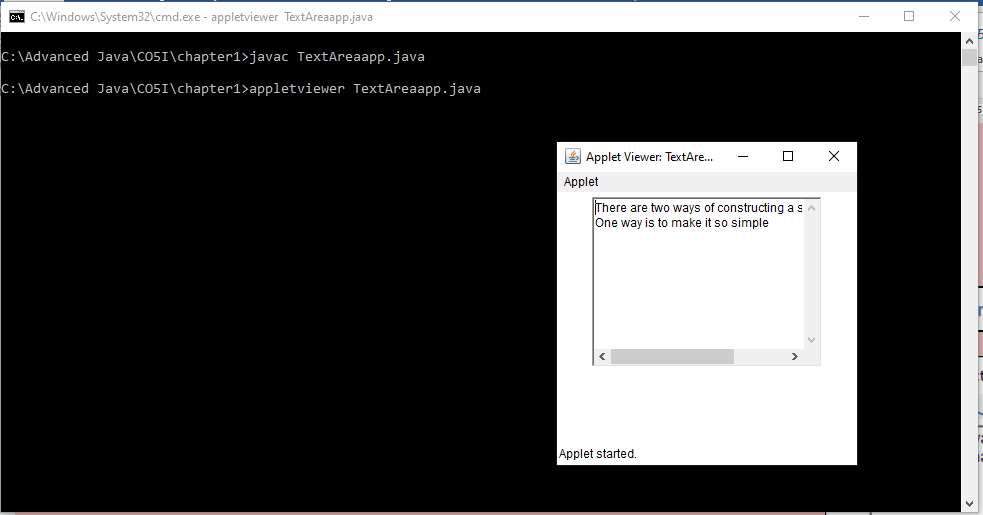
TextArea text = new TextArea(val, 10, 30);

add(text);

}

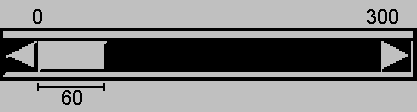
}

No of Lines



**Scroll Bars**

* The [object](https://www.javatpoint.com/object-and-class-in-java) of Scrollbar class is used to add **horizontal and vertical scrollbar.**
* Scrollbar is a [GUI](https://www.javatpoint.com/gui-full-form) component allows us to see invisible number **of rows and columns.**

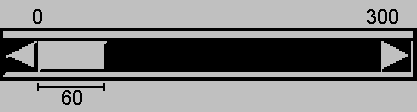


* Scrollbar defines the following constructors:

**Scrollbar(int style**, int initialValue, int thumbSize, int min, int max**)**

for eg.

Scrollbar(**Scrollbar.HORIZONTAL**,0, 60, 0, 300);



**program-**

// Demonstrate scroll bars.

import java.awt.\*;

import java.applet.\*;

/\*

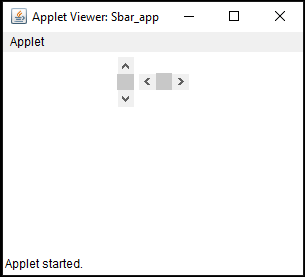
<applet code="Sbar\_app" width=300 height=200>

</applet>

\*/

public class Sbar\_app extends Applet

{

Scrollbar v, h;

public void init()

{

int width = 300;

int height = 200;

v = new Scrollbar(Scrollbar.VERTICAL,0, 100, 0, height);

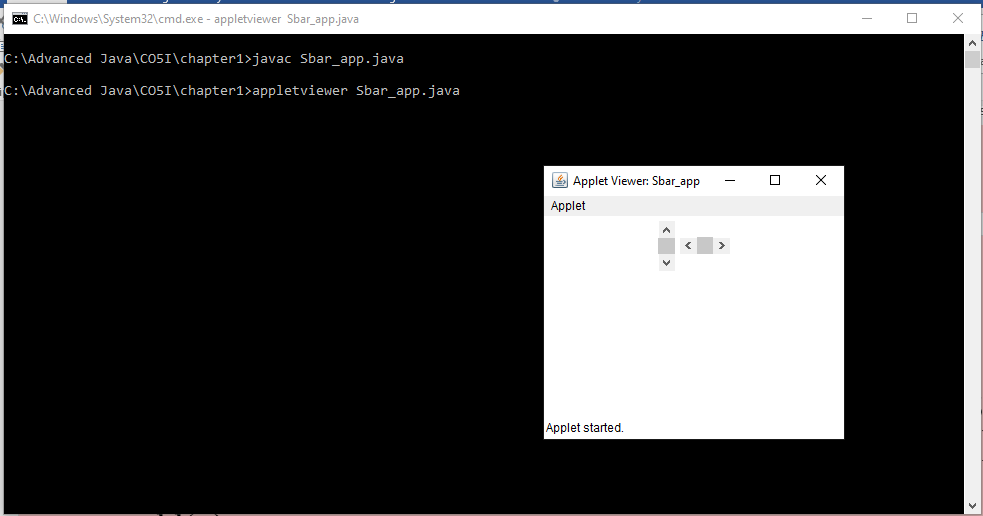
h= new Scrollbar(Scrollbar.HORIZONTAL,10, 1, 0, width);

add(v);

add(h);

}

}



**LayoutManagers**

* The LayoutManagers are used to **arrange components** in a particular manner.
* In other way we can say that placing the components at a **particular position** within the container.
* **LayoutManager** is an **interface** that is implemented by all the classes of layout managers.

**Types of Layout Manager**

* BorderLayout
* FlowLayout
* CardLayout
* GridLayout
* GridbagLayout

# FlowLayout

* The FlowLayout is used to arrange the components **in a line, one after another (in a flow - Left to Right)**.
* It is the **default layout of applet or panel**.
* The layout manager is set by the **setLayout( )**

### Constructors of FlowLayout class

1. **FlowLayout():** flow layout with **centered alignment** and a default **5** unit horizontal and vertical gap.
2. **FlowLayout(int align):**

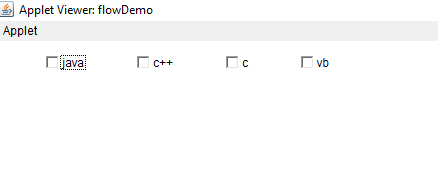
FlowLayout(FlowLayout.LEFT)

1. **FlowLayout(int align, int hgap, int vgap):**

creates a flow layout with the given alignment and the given horizontal and vertical gap.

**for eg.**

FlowLayout(FlowLayout.LEFT,50,10)



**program:**

/\*

import java.awt.\*;

import java.applet.\*;

\*/

<applet code="flowDemo" width=250 height=200>

</applet>

\*/

public class flowDemo extends Applet

{

Checkbox c1,c2,c3,c4;

public void init()

{

FlowLayout f=new FlowLayout(FlowLayout.LEFT);

setLayout(f);

c1 = new Checkbox("java");

c2= new Checkbox("c++");

c3= new Checkbox("c");

c4= new Checkbox("vb");

add(c1);

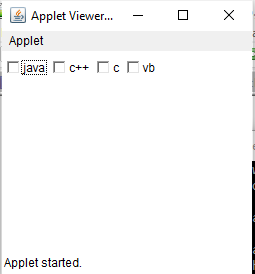
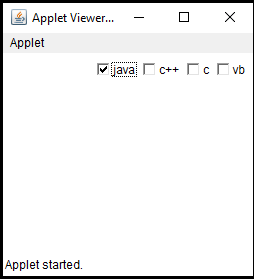
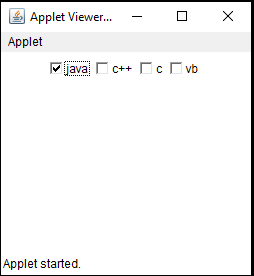
add(c2);

add(c3);

add(c4);

}

}

**BorderLayout**

* The BorderLayout is used to arrange the components in **five regions: north, south, east, west and center**.
* Each region (area) may contain one component only.

It is the **default layout of frame or window**.

* The BorderLayout provides five constants for each region:

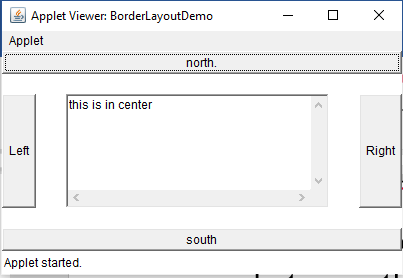
1. **BorderLayout.CENTER**
2. **BorderLayout.SOUTH**
3. **BorderLayout.EAST**
4. **BorderLayout.WEST**
5. **BorderLayout.NORTH**

**Constructors of BorderLayout class:**

* **BorderLayout():** creates a border layout but with **no gaps** between the components.
* **BorderLayout(int hgap, int vgap):**

For eg. BorderLayout(30,20)

* creates a border layout with the given horizontal and vertical gaps between the components.



**program-**

// Demonstrate BorderLayout.

import java.awt.\*;

import java.applet.\*;

/\*

<applet code="BorderLayoutDemo" width=400 height=200>

</applet>

\*/

public class BorderLayoutDemo extends Applet

{

Button b1,b2,b3,b4;

TextArea t1;

String msg = "this is in center";

public void init()

{

setLayout(new BorderLayout());

b1=new Button("north.");

b2=new Button("south");

b3=new Button("Right");

b4=new Button("Left");

t1=new TextArea(msg);

add(b1,BorderLayout.NORTH);

add(b2,BorderLayout.SOUTH);

add(b3, BorderLayout.EAST);

add(b4, BorderLayout.WEST);

add(t1, BorderLayout.CENTER);

}

}

**Same Program:**

// Demonstrate BorderLayout.

import java.awt.\*;

import java.applet.\*;

/\*

<applet code="BorderLayoutDemo" width=400 height=200>

</applet>

\*/

public class BorderLayoutDemo extends Applet

{

public void init()

{

setLayout(new BorderLayout());

add(new Button("north."),BorderLayout.NORTH);

add(new Button("south"),BorderLayout.SOUTH);

add(new Button("Right"), BorderLayout.EAST);

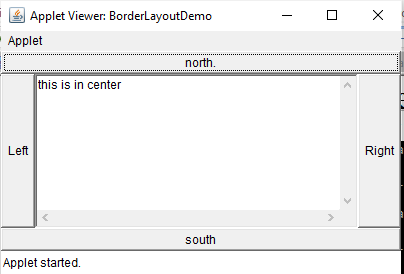
add(new Button("Left"), BorderLayout.WEST);

String msg = "this is in center";

add(new TextArea(msg), BorderLayout.CENTER);

}

}



# GridLayout

* The GridLayout is used to arrange the components in rectangular grid**(matrix).**

### Constructors of GridLayout class

1. **GridLayout():** creates a grid layout with one column and one row.
2. **GridLayout(int rows, int columns):**

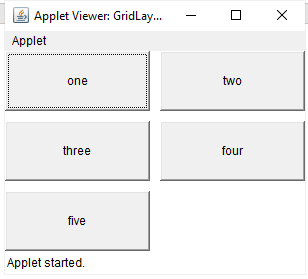
for eg.

**GridLayout(3,2)**

1. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.

For eg.

GridLayout(3, 2,10,10)



**// Demonstrate GridLayout**

**import java.awt.\*;**

**import java.applet.\*;**

**/\***

**<applet code="GridLayoutDemo" width=300 height=200>**

**</applet>**

**\*/**

**public class GridLayoutDemo extends Applet**

**{**

**Button b1,b2,b3,b4,b5,b6;**

**public void init()**

**{**

**b1=new Button("one");**

**b2=new Button("two");**

**b3=new Button("three");**

**b4=new Button("four");**

**b5=new Button("five");**

Rows, columns

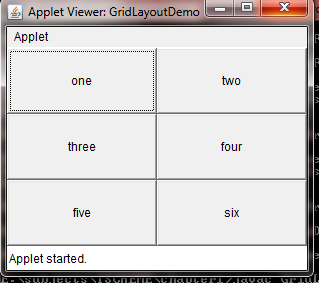
**b6=new Button("six");**

**setLayout(new GridLayout(3,2));**

**add(b1);add(b2);add(b3);add(b4);add(b5);add(b6);**

**}**

**}**

****

**GridBagLayout**

GridBagLayout class is a **flexible layout manager.**

It is used to aligns the components **horizontally, vertically or along their baseline**.

It **doesn’t require the components of the same size.**

Each GridBagLayout object manages a rectangular grid of cells, **dynamic with each component occupying one or more cells, called its display area**.

GridBagLayout components are associated with the instance of **GridBagConstraints**. These constraints are used **to define the component’s display area and their positions**.

**Constructor:**

* **GridBagLayout():** It is used to creates a grid bag layout manager.
* **gridx, gridy**

Specify the row and column at the upper left of the component. The leftmost column has address gridx=0 and the top row has address gridy=0

**gridwidth, gridheight**

Specify the number of columns (for gridwidth) or rows (for gridheight) in the component's display area..

**ipadx, ipady**

* Specifies the internal padding: how much to add to the size of the component. The default value is zero.

.

**insets**

* Specifies the external padding of the component -- the minimum amount of space between the component and the edges of its display area.

**anchor**

* Used when the component is smaller than its display area to determine where (within the area) to place the component. Valid values (defined as GridBagConstraints constants) are CENTER (the default), PAGE\_START, PAGE\_END, LINE\_START, LINE\_END, FIRST\_LINE\_START, FIRST\_LINE\_END, LAST\_LINE\_END, and LAST\_LINE\_START.
* Here is a picture of how these values are interpreted in a container that has the default, left-to-right component orientation.

|  |  |  |
| --- | --- | --- |
| FIRST\_LINE\_START | PAGE\_START | FIRST\_LINE\_END |
| LINE\_START | CENTER | LINE\_END |
| LAST\_LINE\_START | PAGE\_END | LAST\_LINE\_END |

**weightx, weighty**

* Weights are used to determine how to distribute space among columns (weightx) and among rows (weighty); this is important for specifying resizing behavior.

import java.awt.\*;

public class GridBagLayoutExample extends Frame

{

public GridBagLayoutExample()

{

GridBagLayout grid = new GridBagLayout();

GridBagConstraints gbc = new GridBagConstraints

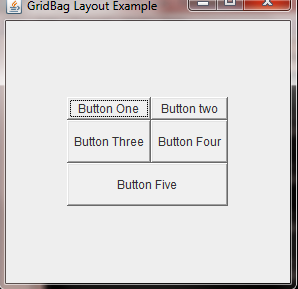
setLayout(grid);

setTitle("GridBag Layout Example");

gbc.fill = GridBagConstraints.HORIZONTAL;

1,0

0,0

 gbc.gridx = 0;

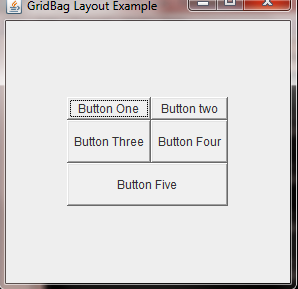
gbc.gridy = 0;

this.add(new Button("Button One"), gbc);

gbc.gridx = 1;

gbc.gridy = 0;

this.add(new Button("Button two"), gbc);

gbc.fill = GridBagConstraints.HORIZONTAL;

gbc.ipady = 20;

gbc.gridx = 0;

gbc.gridy = 1;

this.add(new Button("Button Three"), gbc);

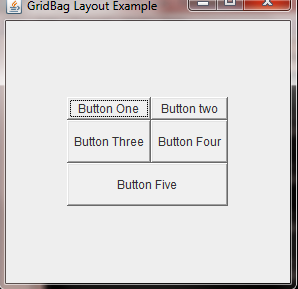
gbc.gridx = 1;

0,1

1,1

gbc.gridy = 1;

this.add(new Button("Button Four"), gbc);

 gbc.fill = GridBagConstraints.HORIZONTAL;

gbc.gridx = 0;

gbc.gridy = 2;

gbc.gridwidth = 2;

this.add(new Button("Button Five"), gbc);

setSize(300, 300);

setVisible(true);

}

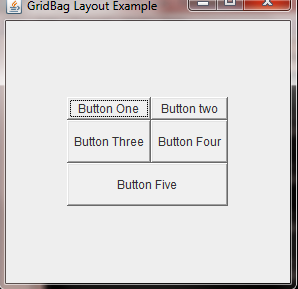
public static void main(String[] args)

{

GridBagLayoutExample a = new GridBagLayoutExample();

}

}



**CardLayout**

* The CardLayout class is unique among the other layout managers in that it **stores several different layouts**

* CardLayout provides these two constructors:

CardLayout( )

CardLayout(int *horz*, int *vert*)

* Use of a card layout requires a bit more work than the other layouts. The cards are typically held in an object of type **Panel**.
* following methods defined by CardLayout:

void first(Container *deck*)

void last(Container *deck*)

void next(Container *deck*)

void previous(Container *deck*)

void show(Container *deck*, String *cardName*)

[CardLayout-2.mp4](file:///C:/Users/SYSTEMS/AppData/Roaming/Microsoft/Word/CardLayout-2.mp4)

import java.awt.\*;

import java.awt.event.\*;

import java.applet.\*;

/\*

<applet code="CardLayoutDemo" width=300 height=100>

</applet>

\*/

public class CardLayoutDemo extends Applet

implements ActionListener

{

Checkbox c1, c2, c3, c4;

Panel p1;

CardLayout c;

Button b1,b2;

public void init()

{

b1 = new Button("last");

b2 = new Button("next");

add(b1);

add(b2);

c = new CardLayout();

p1 = new Panel();

p1.setLayout(c);

// set panel layout to card layout

c1 = new Checkbox("Windows 98/XP");

c2 = new Checkbox("Windows NT/2000");

c3 = new Checkbox("Solaris");

c4 = new Checkbox("MacOS");

// add Windows check boxes to a panel

Panel p2 = new Panel();

p2.add(c1);

p2.add(c2);

// Add other OS check boxes to a panel

Panel p3 = new Panel();

p3.add(c3);

p3.add(c4);

// add panels to card deck panel

p1.add(p2, "second panel");

p1.add(p3, "third panel");

// add cards to main applet panel

add(p1);

// register to receive action events

b1.addActionListener(this);

b2.addActionListener(this);

}

public void actionPerformed(ActionEvent ae)

{

if(ae.getSource() == b1)

{

c.last(p1);

}

else

{

c.next(p1);

}}

}

## MenuBar,

* To create menus, the java.awt package comes with mainly four classes – MenuBar, Menu, MenuItem and CheckboxMenuItem
* **MenuBar:** MenuBar holds the menus. MenuBar is added to frame with **setMenuBar()** method. Implicitly, the menu bar is added to the north (top) of the frame.
* **Menu:** Menu holds the menu for eg File,Edit. Menu is added to frame with **add()**Menu.
* **MenuItem:** MenuItem displays the actual option user can select. Menu items are added to menu with method **addMenuItem()**.
* **CheckboxMenuItem:** It differs from MenuItem in that it appears along with a checkbox. The selection can be done *with checkbox selected.*

//Program

import java.awt.\*;

// Create a subclass of Frame

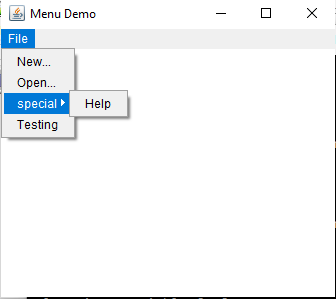
class MenuFrame extends Frame

{

// create menu bar and add it to frame

MenuFrame()

{

MenuBar mbar = new MenuBar();

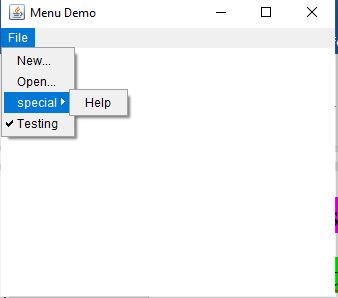
setMenuBar(mbar);

// create the menu items

Menu file = new Menu("File");

MenuItem i1, i2;

file.add(i1 = new MenuItem("New..."));

file.add(i2 = new MenuItem("Open..."));

Menu sub = new Menu(“special");

MenuItem i3 ;

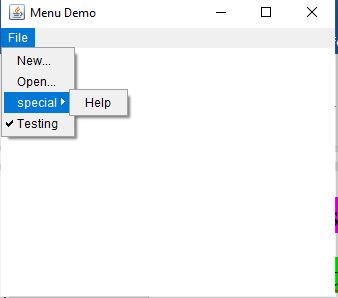
sub.add(i3 = new MenuItem("Help"));

file.add(sub);

// these are checkable menu items

CheckboxMenuItem test = new CheckboxMenuItem("Testing");

file.add(test);

mbar.add(file);

}

public static void main(String args[])

{

MenuFrame f=new MenuFrame();

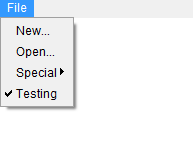
f.setTitle("Menu Demo");

f.setSize(200,200);

f.setVisible(true);

}

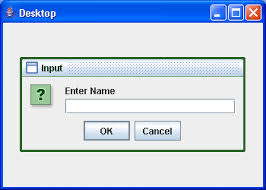
}



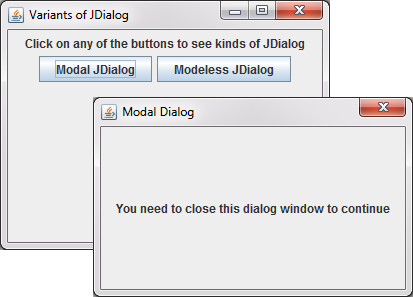
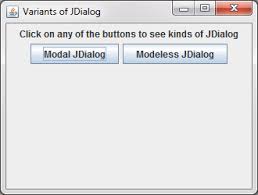
**DialogBox**

Dialog boxes are primarily used **to obtain user input**. They are similar to frame windows, except that **dialog boxes are always child windows of a top-level window**.

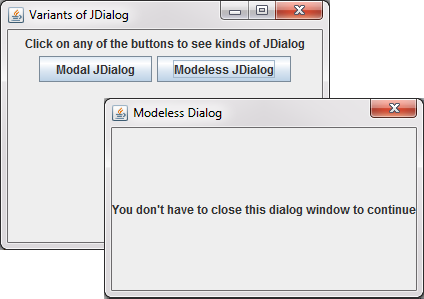
**Dialog** class is used to create a top-level container Dialog window which contains a set of components like button, text field, label, etc.

****

* Dialog boxes may be **modal or modeless**.



* When a ***modal* dialog** box is active,all input is directed to **it until it is closed**. This means that you **cannot access other parts** of your program until you have closed the dialog box.



* When a ***modeless* dialog box** is active, **input focus can be directed** to another window in your program.
* Constructor:

Dialog(Frame *parentWindow*, boolean *mode*)

Dialog(Frame *parentWindow*, String *title*, boolean *mode*)

for eg.

Dialog d = new Dialog(f ,"Dialog Example", **true)**;

if true Dialog box is **modal.**

if false Dialog box is **modeless.**

**FileDialog**

* Java provides a built-in dialog box that lets the user specify a file(open or save).
* provides these constructors:

FileDialog(Frame *parent*, String *boxName*, int *how*)

for eg

FileDialog(Frame *parent*, “**File Dialog”**, **FileDialog.LOAD**)

* Here, *parent* is the owner of the dialog box, and *boxName* is the name displayed in the box’s title bar.
* If *how* is **FileDialog.LOAD**, then the box is selecting a file for **reading(opening any file)**.
* If *how* is **FileDialog.SAVE**,the box is selecting a file for **writing**.

**FileDialog.LOAD(Reading)**



**FileDialog.SAVE(Writing)**