GUI Design

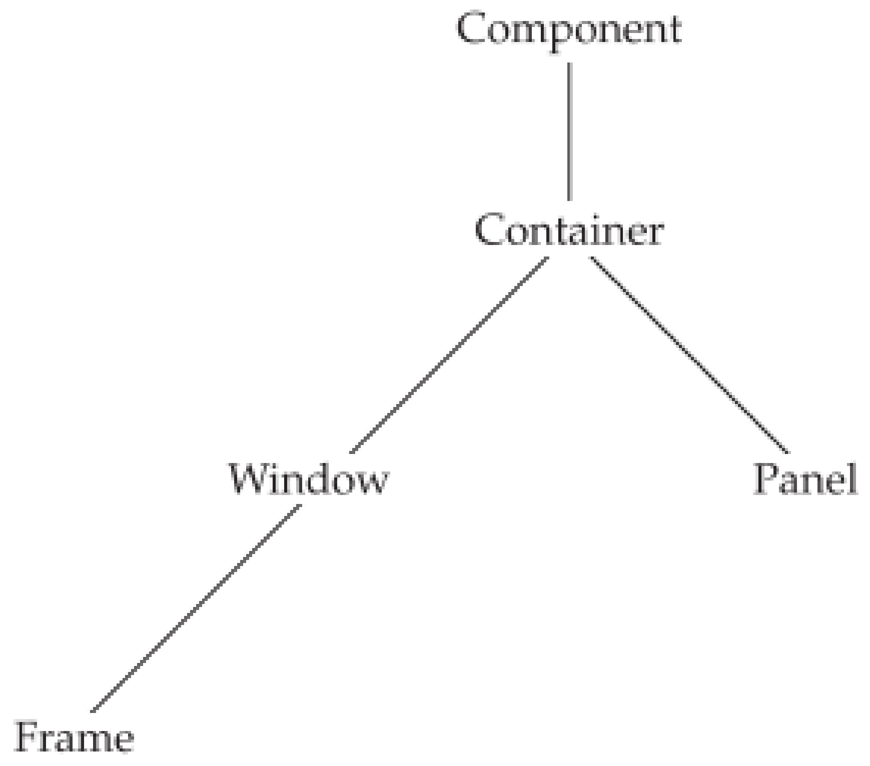
AWT & Swings

AWT

- The AWT (Abstract Window Toolkit) was Java’s first GUI framework, and it has been part of Java since version 1.0.
- It contains numerous classes and methods that allow you to create windows.
- The AWT classes are contained in the java.awt package.
- It is one of Java’s largest packages

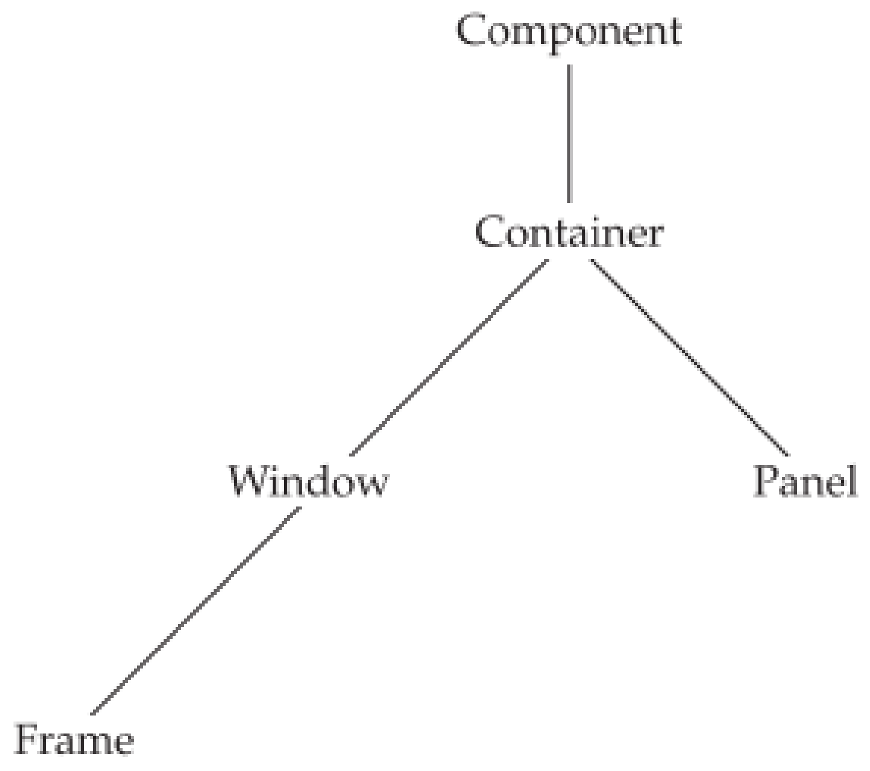
AWT

- 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.
- Much of the functionality of these windows is derived from their parent classes.



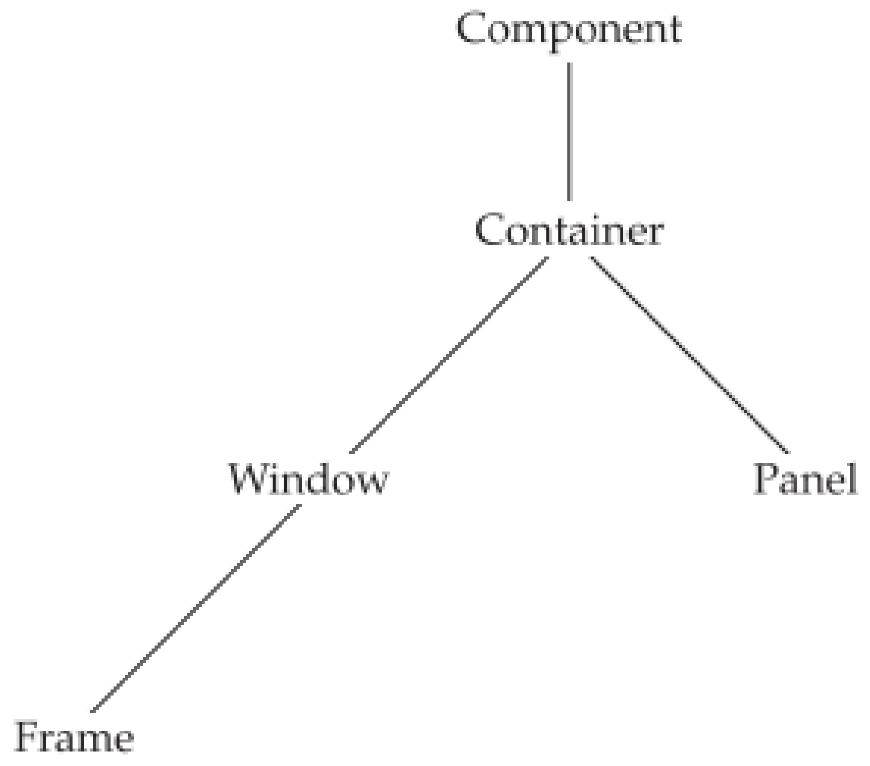
Component

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



Container

- The Container class is a subclass of Component.
- It has additional methods that allow other Component objects to be nested within it. Other Container objects can be stored inside of a Container.



Container

Panel:

- The Panel class is a concrete subclass of Container.
- Panel is the superclass for Applet.
- a Panel is a window that does **not** contain a **title bar**, **menu bar**, or **border**.
- Other components can be added to a Panel object by its add( ) method (inherited from Container).
- Once these components have been added, you can position and resize them manually using the setLocation( ) and setSize( ) methods defined by Component.

Container

Window:

- The Window class creates a top-level window.
- You won’t create Window objects directly. Instead, you will use a subclass of Window called Frame.

Frame:

- Frame encapsulates what is commonly thought of as a **window**.
- It is a subclass of Window and has a **title bar**, **menu bar**, **borders**, and **resizing corners**.

Frame

- You will use it to create child windows.
- Here are two of Frame's constructors:

```
Frame( )
Frame(String title)
```

- The first form creates a standard window that does not contain a title.
- The second form creates a window with the title specified by 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.
- The default layout for a frame is BorderLayout.

Frame

- The setSize( ) method is used to set the dimensions of the window.

```
void setSize(int newWidth, int newHeight)
```

- The new size of the window is specified by newWidth and newHeight.
- The dimensions are specified in terms of pixels.

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

```
Dimension getSize( )
```

- This method returns the current size of the window contained within the width and height

Frame

- After a frame window has been created, it will not be visible until you call setVisible( ).

```
void setVisible(boolean visibleFlag)
```

- The component is visible if the argument to this method is true. Otherwise, it is hidden.

- You can change the title in a frame window using setTitle( ).

```
void setTitle(String newTitle)
```

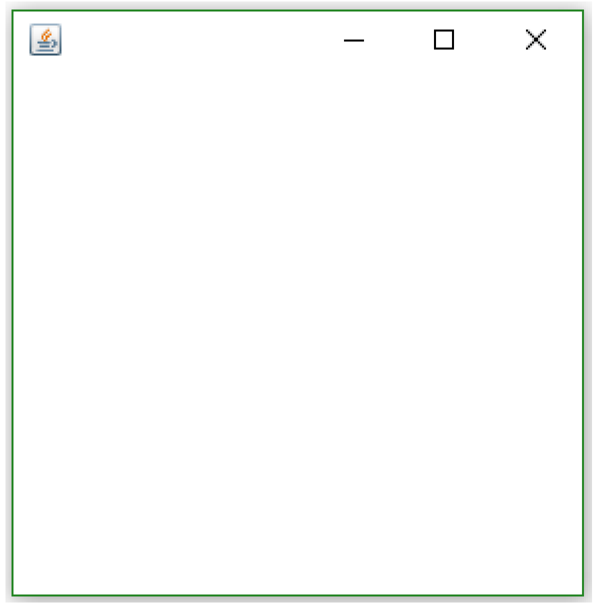
- Here, newTitle is the new title for the window.

Frame

```
import java.awt.*;
public class DemoFrame extends Frame{
        void showFrame()
        {
                setSize(300,300);//frame size 300 width and 300 height
                setVisible(true);//now frame will be visible, by default not visible
        }
  public static void main(String args[]) {
         DemoFrame obj=new DemoFrame();
                obj.showFrame();
  }
}
```

Frame

```
import java.awt.*;
public class DemoFrame extends Frame{
        void showFrame()
        {
                setSize(300,300);//frame size 300 width and 300 height
                setVisible(true);//now frame will be visible, by default not visible
        }
  public static void main(String args[]) {
         DemoFrame obj=new DemoFrame();
                obj.showFrame();
  }
}
```

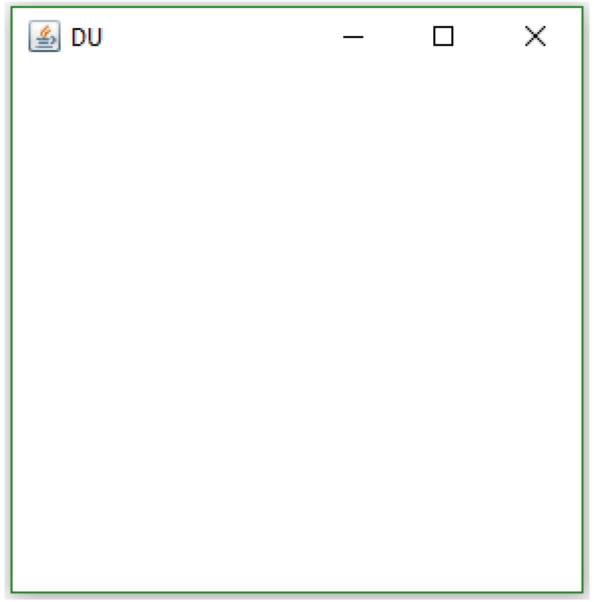


Frame

```
import java.awt.*;
public class DemoFrame extends Frame{
        void showFrame()
        {
                setSize(300,300);//frame size 300 width and 300 height
                setTitle("DU");// now frame has title
                setVisible(true);//now frame will be visible, by default not visible
        }
  public static void main(String args[]) {
         DemoFrame obj=new DemoFrame();
                obj.showFrame();
  }
}
```

Frame

```
import java.awt.*;
public class DemoFrame extends Frame{
            void showFrame()
            {
                        setSize(300,300);//frame size 300 width and 300 height
                        setTitle("DU");// now frame has title
                        setVisible(true);//now frame will be visible, by default not visible
            }
  public static void main(String args[]) {
             DemoFrame obj=new DemoFrame();
                        obj.showFrame();
  }
}
```



Frame

```
import java.awt.*;
public class DemoFrame{
        void showFrame()
        {
                Frame f=new Frame();
                f.setSize(300,300);//frame size 300 width and 300 height
                setTitle(“DU”);// now frame has title
                f.setVisible(true);//now frame will be visible, by default not visible
        }
  public static void main(String args[]) {
         DemoFrame obj=new DemoFrame();
                obj.showFrame();
  }
}
```

AWT Control

- AWT Control Fundamentals.
  - Labels
  - Push buttons
  - Check boxes
  - Choice lists
  - Lists
  - Text Editing
- These controls are subclasses of Component.
- You must add it to the window.
- You must first create an instance of the desired control and then add it to a window by calling add( ) which is defined by Container.

```
Component add(Component compRef)
```

Labels

- A label is an object of type Label, and it contains a string, which it displays.

```
Label( )
Label(String str)
Label(String str, int how)
```

- The first version creates a blank label.
- The second version creates a label that contains the string specified by str. This string is left-justified.
- The third version creates a label that contains the string specified by str using the alignment specified by how.
- The value of how must be one of these three constants: **Label.LEFT**, **Label.RIGHT**, or **Label.CENTER**.

Labels

- You can set or change the text in a label by using the setText( ) method. You can obtain the current label by calling getText( ).

```
void setText(String str)
String getText( )
```

- For setText( ), str specifies the new label.
- For getText( ), the current label is returned.
- You can set the alignment of the string within the label by calling setAlignment( ).

```
void setAlignment(int how)
int getAlignment( )
```

Buttons

- A push button is a component that contains a label and that generates an event when it is pressed. Push buttons are objects of type Button.

```
Button( )
Button(String str)
```

- The first version creates an empty button.
- The second creates a button that contains str as a label.
- After a button has been created, you can set its label by calling setLabel( ).
- You can retrieve its label by calling getLabel( ).

```
void setLabel(String str)
String getLabel( )
```

- Here, str becomes the new label for the button.

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.

```
someComponent.addActionListener(instanceOfMyClass);
```

- That interface defines the actionPerformed( ) method, which is called when an event occurs.

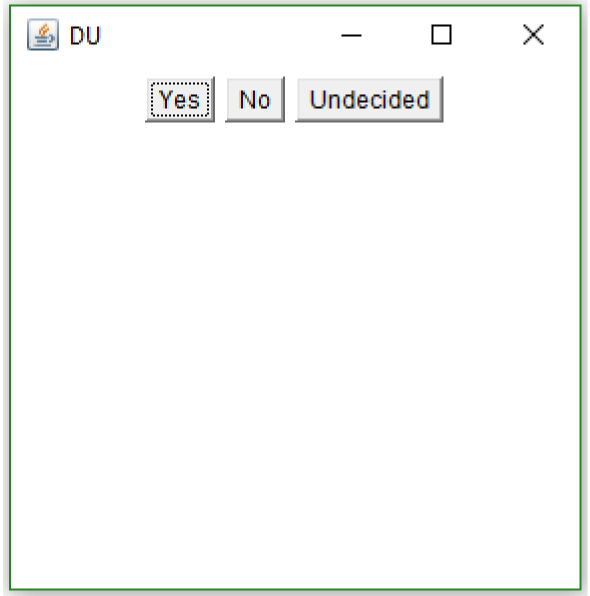
```
public void actionPerformed(ActionEvent e) {
   ...//code that reacts to the action... }
```

- An ActionEvent object is supplied as the argument to this method. It contains both a reference to the button that generated the event and a reference to the action command string associated with the button.

Buttons

```
import java.awt.*;
import java.awt.event.*;
public class DemoFrame extends Frame implements ActionListener {
        Button yes, no, maybe;
        String msg;
        void showFrame()   {
                yes = new Button("Yes");
                no = new Button("No");
                maybe = new Button("Undecided");
                setLayout(new FlowLayout());
                add(yes); add(no);  add(maybe);
                yes.addActionListener(this);
                no.addActionListener(this);
                maybe.addActionListener(this);
                setSize(300,300);
                setTitle("DU");
                setVisible(true);      }
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.";
        }
}
public static void main(String args[]) {
         DemoFrame obj=new DemoFrame();
        obj.showFrame();
  }
}
```

Buttons



Check Boxes

- A check box is a control that is used to turn an option on or off.
- It consists of a small box that can either contain a check mark or not.
- There is a label associated with each check box that describes what option the box represents.
- You change the state of a check box by clicking on it.
- Check boxes can be used individually or as part of a group.

```
Checkbox( )
Checkbox(String str)
Checkbox(String str, boolean on)
Checkbox(String str, boolean on, CheckboxGroup cbGroup)
Checkbox(String str, CheckboxGroup cbGroup, boolean on)
```

Check Boxes

- To retrieve the current state of a check box, call getState( ). To set its state, call setState( ).
- You can obtain the current label associated with a check box by calling getLabel( ).
- To set the label, call setLabel( ).

```
boolean getState( )
void setState(boolean on)
String getLabel( )
void setLabel(String str)
```

- Here, if on is true, the box is checked. If it is false, the box is cleared.

Handling Check Boxes

- Each time a 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.
- Each listener implements the ItemListener interface.
- That interface defines the itemStateChanged( ) method.
- An ItemEvent object is supplied as the argument to this method.

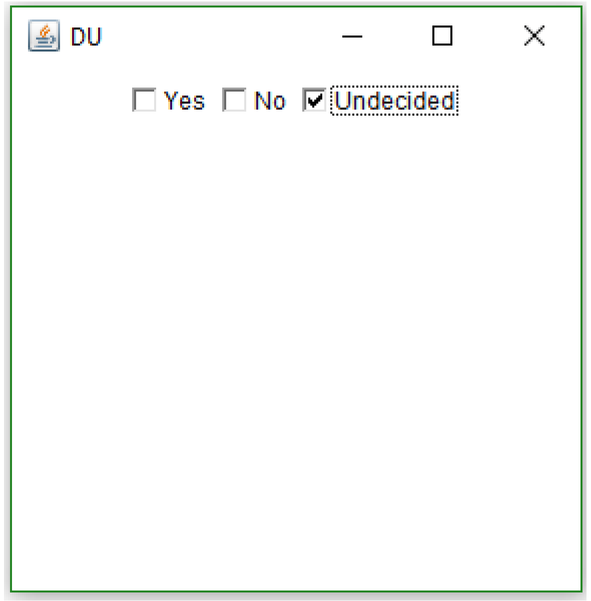
Checkbox

```
import java.awt.*;
import java.awt.event.*;
public class DemoFrame extends Frame implements ItemListener{
        Checkbox yes, no, maybe;
        void showFrame()
        {
                yes = new Checkbox("Yes");
                no = new Checkbox("No");
                maybe = new Checkbox("Undecided");
                setLayout(new FlowLayout());
                add(yes);   add(no);   add(maybe);
                yes.addItemListener(this);
                no.addItemListener(this);
                maybe.addItemListener(this);
                setSize(300,300);
                setTitle("DU");
                setVisible(true)        }
public void itemStateChanged(ItemEvent ae)
 {
                System.out.println(ae);
}
public static void main(String args[]) {
        DemoFrame obj=new DemoFrame();
        obj.showFrame();
  }
}
```

Checkbox

```
import java.awt.*;

import java.awt.event.*;

public    class    DemoFrame    extends    Frame    implements
ItemListener{

        Checkbox yes, no, maybe;

        void showFrame()

        {

                yes = new Checkbox("Yes");

                no = new Checkbox("No");

                maybe = new Checkbox("Undecided");

                setLayout(new FlowLayout());

                add(yes);   add(no);   add(maybe);

                yes.addItemListener(this);

                no.addItemListener(this);

                maybe.addItemListener(this);

                setSize(300,300);

                setTitle("DU");

                setVisible(true)      }
```

```
public void itemStateChanged(ItemEvent ae)

 {

                System.out.println(ae);

}

public static void main(String args[]) {

        DemoFrame obj=new DemoFrame();

        obj.showFrame();

  }

}
```



```
java.awt.event.ItemEvent[ITEM_STATE_CHANGED,item=Undecided,
stateChange=SELECTED] on checkbox0
```

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 often called radio buttons.

CheckboxGroup

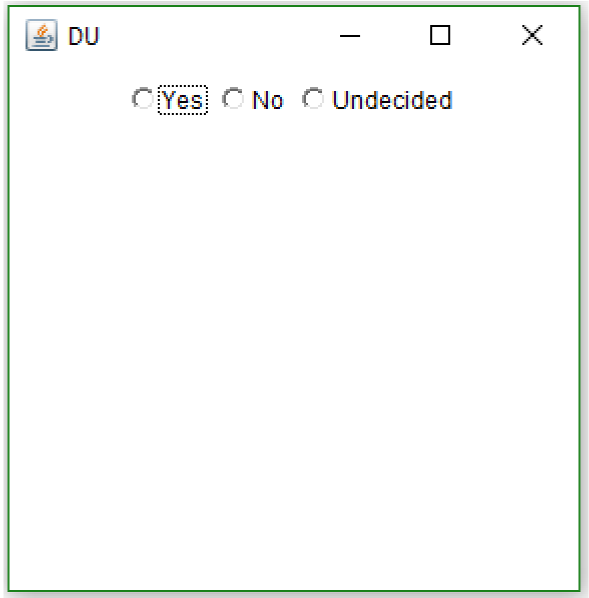
```
import java.awt.*;
import java.awt.event.*;
public class DemoFrame extends Frame implements ItemListener{
        Checkbox yes, no, maybe;
        void showFrame()
        {
        cbg = new CheckboxGroup();
        yes = new Checkbox("Yes", cbg, false);
        no = new Checkbox("No", cbg, false);
        maybe = new Checkbox("Undecided", cbg, false);
                setLayout(new FlowLayout());
                add(yes);   add(no);   add(maybe);
                yes.addItemListener(this);
                no.addItemListener(this);
                maybe.addItemListener(this);
                setSize(300,300);
                setTitle("DU");
```

```
public void itemStateChanged(ItemEvent ae)
 {
                System.out.println(ae);
}
public static void main(String args[]) {
        DemoFrame obj=new DemoFrame();
        obj.showFrame();
  }
}
```

CheckboxGroup

```
import java.awt.*;
import java.awt.event.*;
public class DemoFrame extends Frame implements ItemListener{
        Checkbox yes, no, maybe;
        void showFrame()
        {
        cbg = new CheckboxGroup();
        yes = new Checkbox("Yes", cbg, false);
        no = new Checkbox("No", cbg, false);
        maybe = new Checkbox("Undecided", cbg, false);
                setLayout(new FlowLayout());
                add(yes);   add(no);   add(maybe);
                yes.addItemListener(this);
                no.addItemListener(this);
                maybe.addItemListener(this);
                setSize(300,300);
                setTitle("DU");
```

```
public void itemStateChanged(ItemEvent ae)
 {
                System.out.println(ae);
}
public static void main(String args[]) {
        DemoFrame obj=new DemoFrame();
        obj.showFrame();
  }
}
```



```
java.awt.event.ItemEvent[ITEM_STATE_CHANGED,item=Undecided,
stateChange=SELECTED] on checkbox0
```

Choice Controls

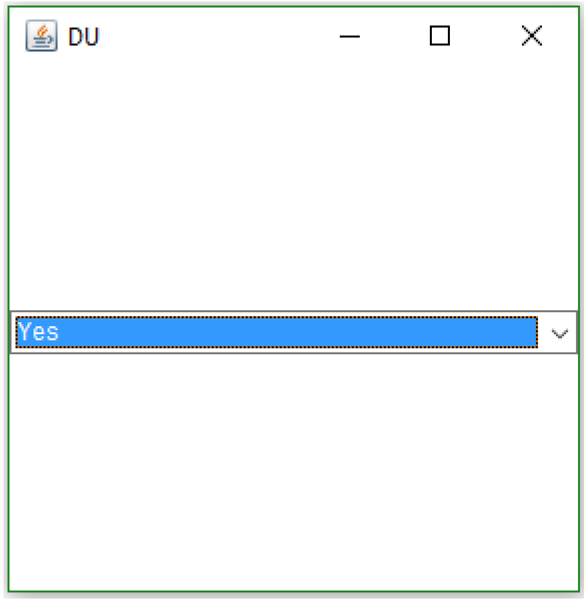
- The Choice class is used to create a pop-up list of items from which the user may choose.
- Thus, a Choice control is a form of menu.
- When the user clicks on it, the whole list of choices pops up, and a new selection can be made.
- Each item in the list is a string that appears as a left-justified label in the order it is added to the Choice object.
- Choice defines only the default constructor, which creates an empty list.

Choice Controls

```
import java.awt.*;

import java.awt.event.*;

public class DemoFrame extends Frame implements ItemListener{

        Choice chs;

        void showFrame()

        {

                chs = new Choice();

                chs.add("Yes");

                chs.add("No");

                chs.add("Maybe");

                setLayout(new FlowLayout());

                add(chs);

                chs.addItemListener(this);
                setSize(300,300);

                setTitle("DU");

                setVisible(true)

        }

public void itemStateChanged(ItemEvent ae)

{

                System.out.println(ae);

}

public static void main(String args[]) {

        DemoFrame obj=new DemoFrame();

        obj.showFrame();

  }

}
```

Choice Controls

```
import java.awt.*;

import java.awt.event.*;

public class DemoFrame extends Frame implements ItemListener{

        Choice chs;

        void showFrame()

        {

                chs = new Choice();

                chs.add("Yes");

                chs.add("No");

                chs.add("Maybe");

                setLayout(new FlowLayout());

                add(chs);

                chs.addItemListener(this);
                setSize(300,300);

                setTitle("DU");

                setVisible(true)

        }

public void itemStateChanged(ItemEvent ae)

 {

                System.out.println(ae);

}

public static void main(String args[]) {

                DemoFrame obj=new DemoFrame();

                obj.showFrame();

  }

}
```



```
java.awt.event.ItemEvent[ITEM_STATE_CHANGED,item=No,
stateChange=SELECTED] on choice0
```

Lists

- The List class provides a compact, multiple-choice, scrolling selection list.
- Unlike the Choice object, which shows only the single selected item in the menu, a List object can be constructed to show any number of choices in the visible window.
- It can also be created to allow multiple selections.

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

- Here, name is the name of the item added to the list. The first form adds items to the end of the list.
- The second form adds the item at the index specified by index. Indexing begins at zero.

Lists

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

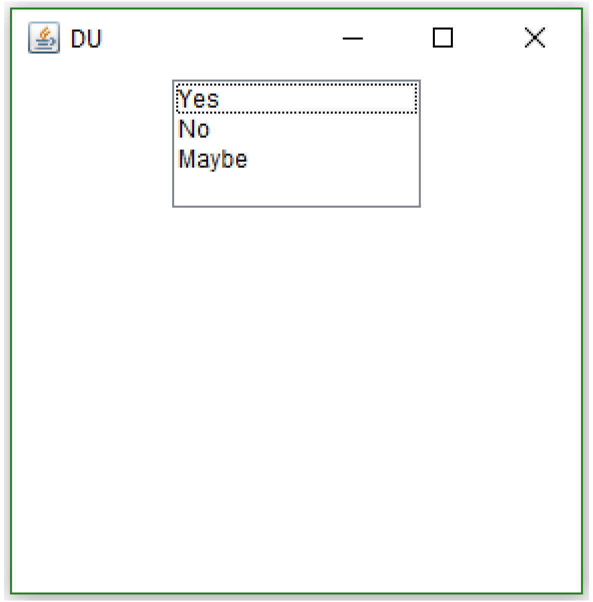
- The first version creates a List control that allows only one item to be selected at any one time.
- In the second form, the value of numRows specifies the number of entries in the list that will always be visible (others can be scrolled into view as needed).
- In the third form, if multipleSelect is true, then the user may select two or more items at a time. If it is false, then only one item may be selected

Lists

```
import java.awt.*;
import java.awt.event.*;
public class DemoFrame extends Frame implements ItemListener{
        List chs;
        void showFrame()
        {
                chs = new List();
                chs.add("Yes");
                chs.add("No");
                chs.add("Maybe");
                setLayout(new FlowLayout());
                add(chs);
                chs.addItemListener(this);
                setSize(300,300);
                setTitle("DU");
                setVisible(true)
        }
public void itemStateChanged(ItemEvent ae)
 {
                System.out.println(ae);
}
public static void main(String args[]) {
        DemoFrame obj=new DemoFrame();
        obj.showFrame();
  }
}
```

Lists

```
import java.awt.*;

import java.awt.event.*;

public class DemoFrame extends Frame implements ItemListener{

        List chs;

        void showFrame()

        {

                chs = new List();

                chs.add("Yes");

                chs.add("No");

                chs.add("Maybe");

                setLayout(new FlowLayout());

                add(chs);

                chs.addItemListener(this);
                setSize(300,300);

                setTitle("DU");

                setVisible(true)

        }
public void itemStateChanged(ItemEvent ae)

 {

                System.out.println(ae);

}

public static void main(String args[]) {

        DemoFrame obj=new DemoFrame();

        obj.showFrame();

  }

}
```



```
java.awt.event.ItemEvent[ITEM_STATE_CHANGED,item=1,
stateChange=SELECTED] on list0
```

TextField

- The TextField class implements a single-line text-entry area, usually called an edit control.
- Text fields allow the user to enter strings and to edit the text using the arrow keys, cut and paste keys, and mouse selections.
- TextField is a subclass of TextComponent.

```
TextField( )
TextField(int numChars)
TextField(String str)
TextField(String str, int numChars)
```

- The first version creates a default text field.
- The second creates a text field that is numChars characters wide.
- The third form initializes the text field with the string contained in str.
- The fourth form initializes a text field and sets its width.

TextField

- To obtain the string currently contained in the text field, call getText().
- To set the text, call setText( ).

```
String getText( )
void setText(String str)
```

- Your program can obtain the currently selected text by calling getSelectedText( ).

```
String getSelectedText( )
```

TextField

- There may be times when you will want the user to enter text that is not displayed, such as a password.
- You can disable the echoing of the characters as they are typed by calling setEchoChar( ).
- This method specifies a single character that the TextField will display when characters are entered (thus, the actual characters typed will not be shown)

```
void setEchoChar(char ch)
```

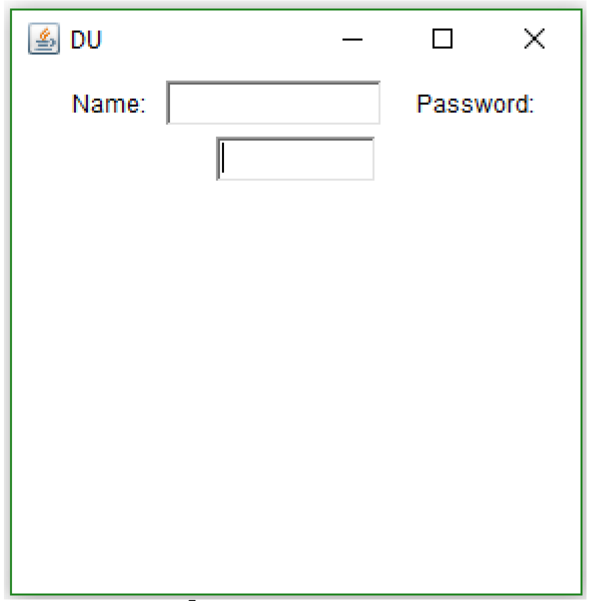
TextField

```
import java.awt.*;
import java.awt.event.*;
public class DemoFrame extends Frame implements ActionListener{
        TextField name, pass;
        void showFrame()
        {
        Label namep = new Label("Name: ", Label.RIGHT);
        Label passp = new Label("Password: ", Label.RIGHT);
                name = new TextField(12);
                pass = new TextField(8);
                pass.setEchoChar('?');
                setLayout(new FlowLayout());
                add(namep);         add(name);
                add(passp);         add(pass);
                name.addActionListener(this);
                pass.addActionListener(this);
        setSize(300,300);  setTitle("DU");  setVisible(true); }
public void actionPerformed(ActionEvent ae)
 {
                System.out.println(ae);
}
public static void main(String args[]) {
        DemoFrame obj=new DemoFrame();
        obj.showFrame();
  }
}
```

TextField

```
import java.awt.*;

import java.awt.event.*;

public    class    DemoFrame    extends    Frame    implements
ActionListener{

        TextField name, pass;

        void showFrame()

        {

        Label namep = new Label("Name: ", Label.RIGHT);

        Label passp = new Label("Password: ", Label.RIGHT);

                name = new TextField(12);

                pass = new TextField(8);

                pass.setEchoChar('?');

                setLayout(new FlowLayout());

                add(namep);         add(name);

                add(passp);         add(pass);

                name.addActionListener(this);

                pass.addActionListener(this);
        setSize(300,300);  setTitle("DU");  setVisible(true); }
```

```
public void actionPerformed(ActionEvent ae)

 {

                        System.out.println(ae);

}

public static void main(String args[]) {

          DemoFrame obj=new DemoFrame();

         obj.showFrame();

  }

}
```



```
java.awt.event.ActionEvent[ACTION_PERFORMED,cmd=ram,when=1625545
088170,modifiers=] on textfield0
java.awt.event.ActionEvent[ACTION_PERFORMED,cmd=123,when=1625545
097893,modifiers=] on textfield1
```

TextArea

- Sometimes a single line of text input is not enough for a given task. To handle these situations, the AWT includes a simple multiline editor called TextArea.

```
TextArea( )
TextArea(int numLines, int numChars)
TextArea(String str)
TextArea(String str, int numLines, int numChars)
TextArea(String str, int numLines, int numChars, int sBars)
```

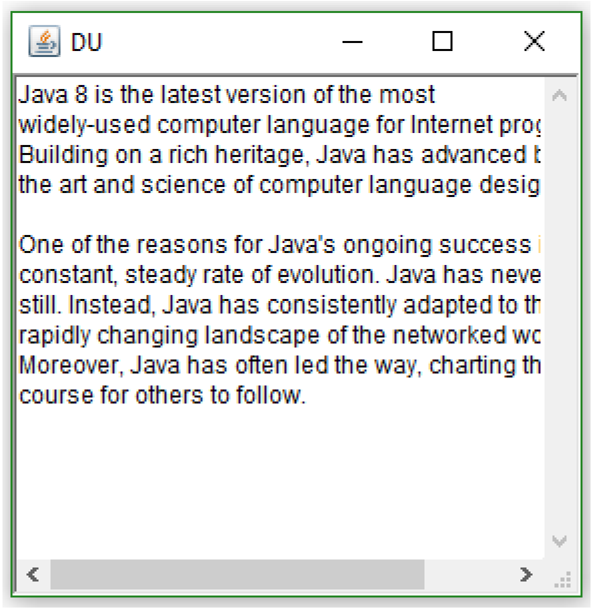
- Here, numLines specifies the height, in lines, of the text area, and numChars specifies its width, in characters.
- Initial text can be specified by str.
- In the fifth form, you can specify the scroll bars that you want the control to have. sBars must be one of these values:
- SCROLLBARS_BOTH SCROLLBARS_NONE
- SCROLLBARS_HORIZONTAL_ONLY SCROLLBARS_VERTICAL_ONLY

TextArea

```
import java.awt.*;
import java.awt.event.*;
public class DemoFrame extends Frame implements {
                 TextField name, pass;
                void showFrame()
                {
                                String val =
"Java 8 is the latest version of the most\n" +
"widely-used computer language for Internet programming.\n" +
"Building on a rich heritage, Java has advanced both\n" +
"the art and science of computer language design.\n\n" +
"One of the reasons for Java's ongoing success is its.";
                TextArea text = new TextArea(val, 10, 30);
                add(text);
                setSize(300,300);
                setTitle("DU");
                setVisible(true); }
public static void main(String args[]) {
                 DemoFrame obj=new DemoFrame();
                obj.showFrame();
   }
}
```

TextArea

```
import java.awt.*;
import java.awt.event.*;
public class DemoFrame extends Frame implements {
                TextField name, pass;
               void showFrame()
               {
                              String val =
"Java 8 is the latest version of the most\n" +
"widely-used computer language for Internet programming.\n" +
"Building on a rich heritage, Java has advanced both\n" +
"the art and science of computer language design.\n\n" +
"One of the reasons for Java's ongoing success is its.";
               TextArea text = new TextArea(val, 10, 30);
               add(text);
               setSize(300,300);
               setTitle("DU");
               setVisible(true); }
public static void main(String args[]) {
                DemoFrame obj=new DemoFrame();
               obj.showFrame();
  }
}
```



```
Java 8 is the latest version of the most
widely-used computer language for Internet prog
Building on a rich heritage, Java has advanced b
the art and science of computer language desig

One of the reasons for Java's ongoing success i
constant, steady rate of evolution. Java has neve
still. Instead, Java has consistently adapted to th
rapidly changing landscape of the networked wo
Moreover, Java has often led the way, charting th
course for others to follow.
```

Layout Managers

- The LayoutManagers are used to arrange components in a particular manner.
- LayoutManager is an interface that is implemented by all the classes of layout managers.
    - FlowLayout
    - BorderLayout
    - GridLayout
    - CardLayout
    - GridBagLayout
- Each Container object has a layout manager associated with it.
- The layout manager is set by the **setLayout()** method.
- If no call to setLayout( ) is made, then the default layout manager is used.

```
void setLayout(LayoutManager layoutObj)
```

Layout Managers

- If you wish to disable the layout manager and position components manually, pass **null** for *layoutObj*.
- If you do this, you will need to determine the shape and position of each component manually, using the setBounds( ) method defined by Component.
    - public void setBounds(int x, int y, int width, int height)
- This puts the upper left corner at location (x, y), where x the the number of pixels from the left of the screen and y is is the number from the top of the screen.

FlowLayout

- FlowLayout is the default layout manager.
- FlowLayout implements a simple layout style, which is similar to how words flow in a text editor.
- The direction of the layout is governed by the container’s component orientation property, which, by default, is left to right, top to bottom.
- A small space is left between each component, above and below, as well as left and right.

FlowLayout

- FlowLayout( )
- FlowLayout(int how)
- FlowLayout(int how, int horz, int vert)

- The first form creates the default layout, which centers components and leaves five pixels of space between each component.
- The second form lets you specify how each line is aligned.
  - FlowLayout.LEFT
  - FlowLayout.CENTER
  - FlowLayout.RIGHT
  - FlowLayout.LEADING
  - FlowLayout.TRAILING

The third constructor allows you to specify the horizontal and vertical space left between components in horz and vert, respectively.

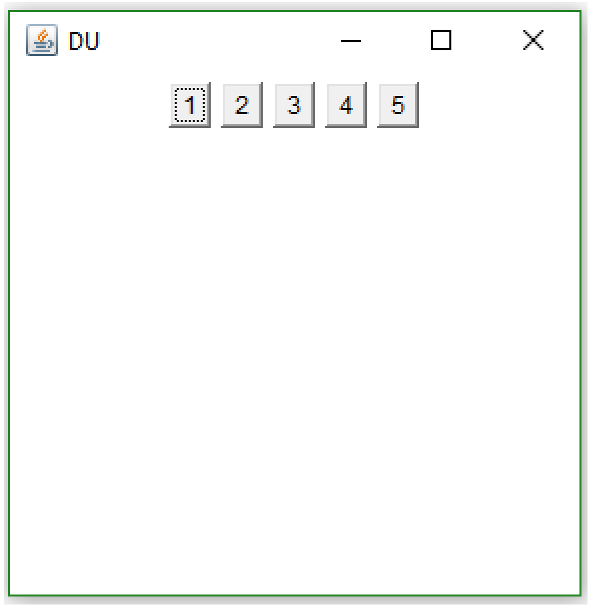
void add(Component compRef, Object region)

FlowLayout

```
import java.awt.*;
public class Demo extends Frame {
	void showFrame(){
			setLayout(new FlowLayout());
			Button b1=new Button("1");
			Button b2=new Button("2");
			Button b3=new Button("3");
			Button b4=new Button("4");
			Button b5=new Button("5");
			add(b1,  FlowLayout.LEFT);
			add(b2, FlowLayout.CENTER);
			add(b3, FlowLayout.RIGHT);
			add(b4, FlowLayout.LEADING);
			add(b5, FlowLayout.TRAILING);
			setSize(300,300);
			setTitle("DU");
			setVisible(true);
	} public static void main(String args[]) {
	  Demo obj=new Demo();
			obj.showFrame();
	 }}
```

FlowLayout

```
import java.awt.*;
public class Demo extends Frame {
	void showFrame(){
			setLayout(new FlowLayout());
			Button b1=new Button("1");
			Button b2=new Button("2");
			Button b3=new Button("3");
			Button b4=new Button("4");
			Button b5=new Button("5");
			add(b1,  FlowLayout.LEFT);
			add(b2, FlowLayout.CENTER);
			add(b3, FlowLayout.RIGHT);
			add(b4, FlowLayout.LEADING);
			add(b5, FlowLayout.TRAILING);
			setSize(300,300);
			setTitle("DU");
			setVisible(true);
	} public static void main(String args[]) {
	  Demo obj=new Demo();
			obj.showFrame();
	 }}
```



BorderLayout

- The BorderLayout class implements a common layout style for top-level windows.
- It has four narrow, fixed-width components at the edges and one large area in the center.
- The four sides are referred to as north, south, east, and west.
- The middle area is called the center.
    - BorderLayout( )
    - BorderLayout(int horz, int vert)
- The first form creates a default border layout.
- The second allows you to specify the horizontal and vertical space left between components in horz and vert, respectively.

BorderLayout

- BorderLayout.CENTER
- BorderLayout.SOUTH
- BorderLayout.EAST
- BorderLayout.WEST
- BorderLayout.NORTH

- The first form creates a default border layout.
- The second allows you to specify the horizontal and vertical space left between components in horz and vert, respectively.

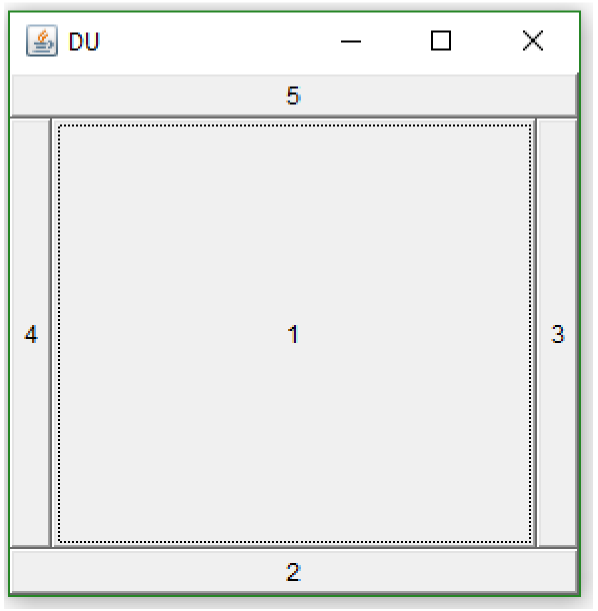
```
void add(Component compRef, Object region)
```

BorderLayout

```
import java.awt.*;
public class Demo extends Frame {
        void showFrame(){
                setLayout(new BorderLayout());
                Button b1=new Button("1");
                Button b2=new Button("2");
                Button b3=new Button("3");
                Button b4=new Button("4");
                Button b5=new Button("5");
                add(b1, BorderLayout.CENTER);
                add(b2, BorderLayout.SOUTH);
                add(b3, BorderLayout.EAST);
                add(b4, BorderLayout.WEST);
                add(b5, BorderLayout.NORTH);
                setSize(300,300);
                setTitle("DU");
                setVisible(true);
        } public static void main(String args[]) {
          Demo obj=new Demo();
                obj.showFrame();
         }}
```

BorderLayout

```
import java.awt.*;
public class Demo extends Frame {
        void showFrame(){
                        setLayout(new BorderLayout());
                        Button b1=new Button("1");
                        Button b2=new Button("2");
                        Button b3=new Button("3");
                        Button b4=new Button("4");
                        Button b5=new Button("5");
                        add(b1, BorderLayout.CENTER);
                        add(b2, BorderLayout.SOUTH);
                        add(b3, BorderLayout.EAST);
                        add(b4, BorderLayout.WEST);
                        add(b5, BorderLayout.NORTH);
                        setSize(300,300);
                        setTitle("DU");
                        setVisible(true);
        } public static void main(String args[]) {
           Demo obj=new Demo();
                        obj.showFrame();
          }}
```



GridLayout

- GridLayout lays out components in a two-dimensional grid. When you instantiate a GridLayout, you define the number of rows and columns.
    - GridLayout( )
    - GridLayout(int numRows, int numColumns)
    - GridLayout(int numRows, int numColumns, int horz, int vert)
- The first form creates a single-column grid layout.
- The second form creates a grid layout with the specified number of rows and columns.
- The third form allows you to specify the horizontal and vertical space left between components in horz and vert, respectively.
- Either numRows or numColumns can be zero.
- Specifying numRows as zero allows for unlimitedlength columns.
- Specifying numColumns as zero allows for unlimited-length rows.

GridLayout

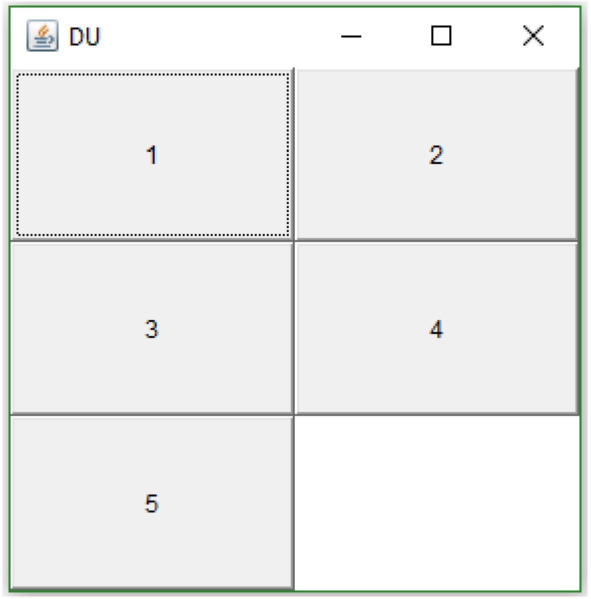
- GridLayout lays out components in a two-dimensional grid. When you instantiate a GridLayout, you define the number of rows and columns.
    - GridLayout( )
    - GridLayout(int numRows, int numColumns)
    - GridLayout(int numRows, int numColumns, int horz, int vert)
- The first form creates a single-column grid layout.
- The second form creates a grid layout with the specified number of rows and columns.
- The third form allows you to specify the horizontal and vertical space left between components in horz and vert, respectively.
- Either numRows or numColumns can be zero.
- Specifying numRows as zero allows for unlimitedlength columns.
- Specifying numColumns as zero allows for unlimited-length rows.

GridLayout

```
import java.awt.*;
public class Demo extends Frame {
        void showFrame(){
                         setLayout(new GridLayout(3,3));
                        Button b1=new Button("1");
                        Button b2=new Button("2");
                        Button b3=new Button("3");
                        Button b4=new Button("4");
                        Button b5=new Button("5");
                                                        add(b1);
                                                        add(b2);
                                                        add(b3);
                                                        add(b4);
                                                        add(b5);
                        setSize(300,300);
                        setTitle("DU");
                        setVisible(true);
        } public static void main(String args[]) {
           Demo obj=new Demo();
                        obj.showFrame();
          }}
```

GridLayout

```
import java.awt.*;
public class Demo extends Frame {
      void showFrame(){
                   setLayout(new GridLayout(3,3));
                  Button b1=new Button("1");
                  Button b2=new Button("2");
                  Button b3=new Button("3");
                  Button b4=new Button("4");
                  Button b5=new Button("5");
                                          add(b1);
                                          add(b2);
                                          add(b3);
                                          add(b4);
                                          add(b5);
                  setSize(300,300);
                  setTitle("DU");
                  setVisible(true);
      } public static void main(String args[]) {
        Demo obj=new Demo();
                  obj.showFrame();
       }}
```



CardLayout

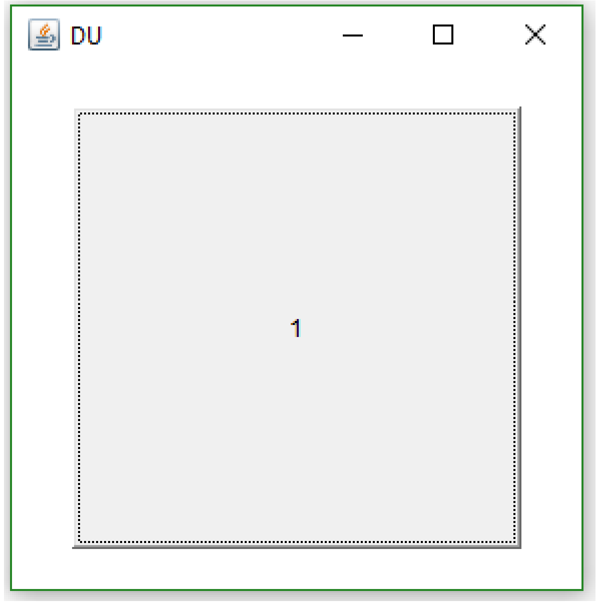
- The CardLayout class manages the components in such a manner that only one component is visible at a time.
- It treats each component as a card that is why it is known as CardLayout.
    - CardLayout( )
    - CardLayout(int horz, int vert)
- The first form creates a default card layout.
- The second form allows you to specify the horizontal and vertical space left between components in horz and vert, respectively.

CardLayout

```
import java.awt.*;
public class Demo extends Frame {
        void showFrame(){
                 setLayout(new CardLayout(3,3));
                Button b1=new Button("1");
                Button b2=new Button("2");
                Button b3=new Button("3");
                Button b4=new Button("4");
                Button b5=new Button("5");
                                add(b1);
                                add(b2);
                                add(b3);
                                add(b4);
                                add(b5);
                setSize(300,300);
                setTitle("DU");
                setVisible(true);
        } public static void main(String args[]) {
          Demo obj=new Demo();
                obj.showFrame();
         }}
```

CardLayout

```
import java.awt.*;
public class Demo extends Frame {
      void showFrame(){
                   setLayout(new CardLayout(3,3));
                  Button b1=new Button("1");
                  Button b2=new Button("2");
                  Button b3=new Button("3");
                  Button b4=new Button("4");
                  Button b5=new Button("5");
                                        add(b1);
                                        add(b2);
                                        add(b3);
                                        add(b4);
                                        add(b5);
                  setSize(300,300);
                  setTitle("DU");
                  setVisible(true);
      } public static void main(String args[]) {
        Demo obj=new Demo();
                  obj.showFrame();
       }}
```



GridBagLayout

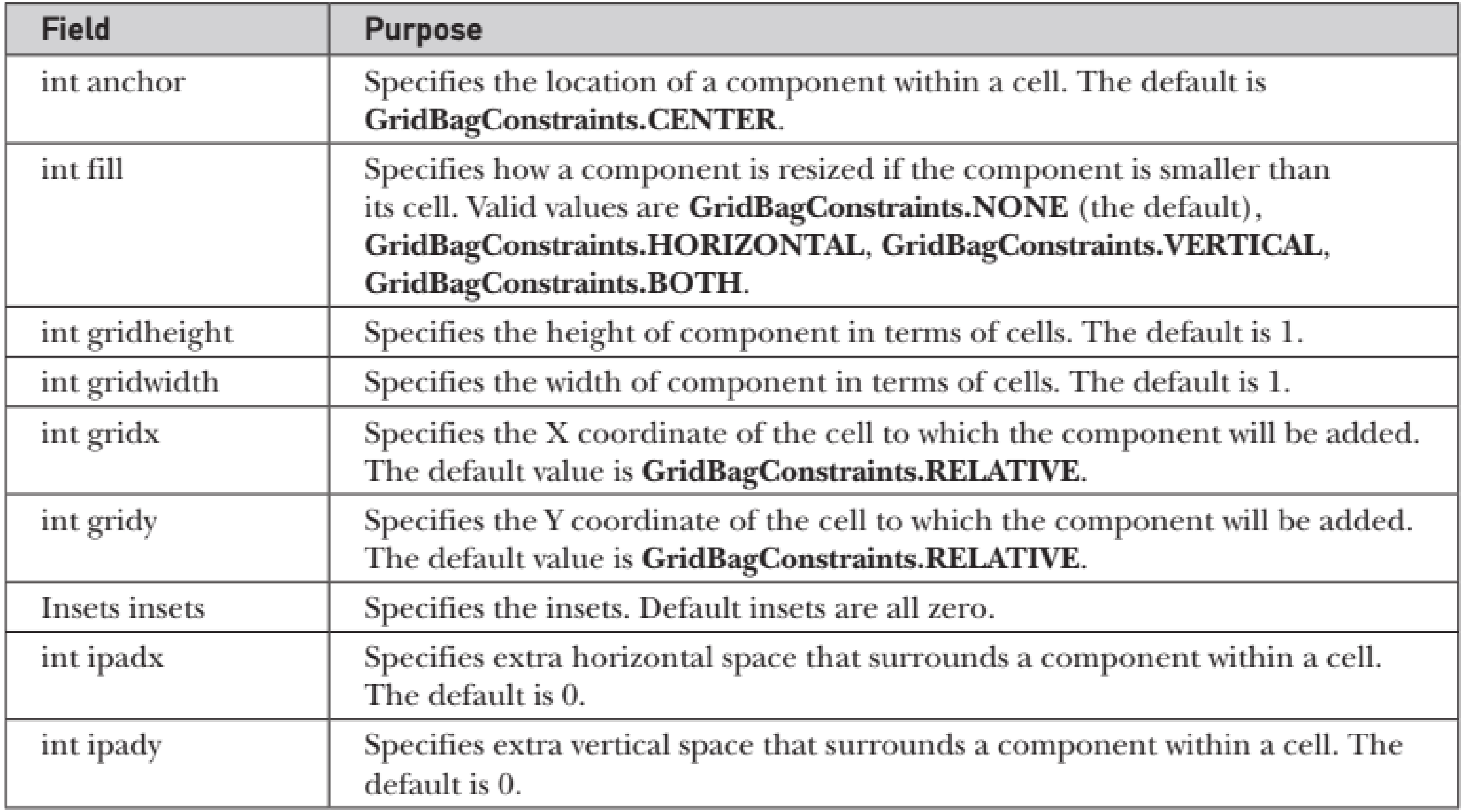
- GridBagLayout is one of the most flexible and complex layout managers the Java platform provides.
- A GridBagLayout places components in a grid of rows and columns, allowing specified components to span multiple rows or columns.
- Not all rows necessarily have the same height.
- Similarly, not all columns necessarily have the same width.
- Each component associates an instance of **GridBagConstraints**. With the help of constraints object we arrange component's display area on the grid.
- The GridBagConstraints class specifies constraints for components that are laid out using the GridBagLayout class.

GridBagLayout

- The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.
    - GridBagLayout( )
- GridBagLayout defines several methods, of which many are protected and not for general use.
- There is one method, however, that you must use: setConstraints( ).
    - void setConstraints(Component comp, GridBagConstraints cons)
- Here, comp is the component for which the constraints specified by cons apply.
- This method sets the constraints that apply to each component in the grid bag.

GridBagLayout

You can set the following GridBagConstraints instance variables:



Field	Purpose
int anchor	Specifies the location of a component within a cell. The default is **GridBagConstraints.CENTER**.
int fill	Specifies how a component is resized if the component is smaller than its cell. Valid values are **GridBagConstraints.NONE** (the default), **GridBagConstraints.HORIZONTAL**, **GridBagConstraints.VERTICAL**, **GridBagConstraints.BOTH**.
int gridheight	Specifies the height of component in terms of cells. The default is 1.
int gridwidth	Specifies the width of component in terms of cells. The default is 1.
int gridx	Specifies the X coordinate of the cell to which the component will be added. The default value is **GridBagConstraints.RELATIVE**.
int gridy	Specifies the Y coordinate of the cell to which the component will be added. The default value is **GridBagConstraints.RELATIVE.**
Insets insets	Specifies the insets. Default insets are all zero.
int ipadx	Specifies extra horizontal space that surrounds a component within a cell. The default is 0.
int ipady	Specifies extra vertical space that surrounds a component within a cell. The default is 0.

GridBagLayout

- GridBagConstraints also defines several static fields that contain standard constraint.
    - GridBagConstraints.CENTER
    - GridBagConstraints.SOUTH
    - GridBagConstraints.EAST
    - GridBagConstraints.SOUTHEAST
    - GridBagConstraints.NORTH
    - GridBagConstraints.SOUTHWEST
    - GridBagConstraints.NORTHEAST
    - GridBagConstraints.WEST
    - GridBagConstraints.NORTHWEST

GridBagLayout

- GridBagConstraints also defines several static fields that resize the component.
  - GridBagConstraints.BOTH
  - GridBagConstraints.NONE
  - GridBagConstraints.HORIZONTAL
  - GridBagConstraints.VERTICAL
- GridBagConstraints also defines some static fields that place the component.
  - GridBagConstraints.RELATIVE
  - GridBagConstraints.REMAINDER

GridBagLayout

GridBagConstraints()

- Creates a GridBagConstraint object with all of its fields set to their default value.

GridBagConstraints(int gridx, int gridy, int gridwidth, int gridheight, double weightx, double weighty, int anchor, int fill, Insets insets, int ipadx, int ipady)

- Creates a GridBagConstraints object with all of its fields set to the passed-in arguments.

- https://docs.oracle.com/javase/tutorial/uiswing/layout/gridbag.html

GridBagLayout

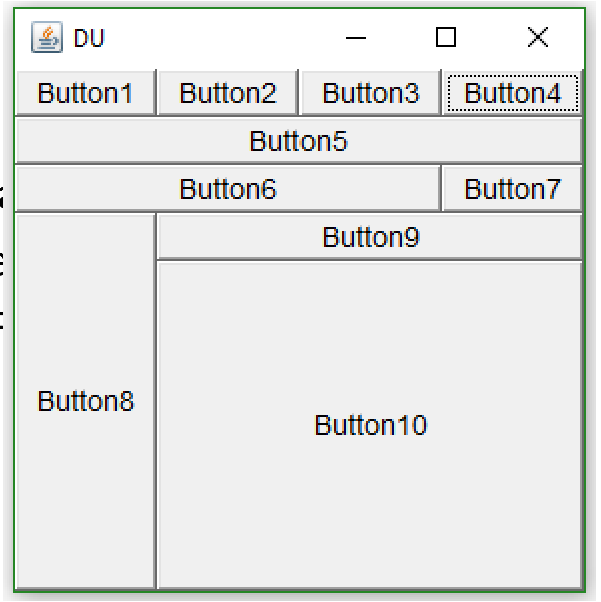
```
void showFrame(){
        GridBagLayout gridbag = new GridBagLayout();
        GridBagConstraints c = new GridBagConstraints();
        setFont(new Font("SansSerif", Font.PLAIN, 14));
        setLayout(gridbag);
        c.fill = GridBagConstraints.BOTH;
        c.weightx = 1.0;
        makebutton("Button1", gridbag, c);
        makebutton("Button2", gridbag, c);
        makebutton("Button3", gridbag, c);
        c.gridwidth = GridBagConstraints.REMAINDER; //end row
        makebutton("Button4", gridbag, c);
        c.weightx = 0.0;             //reset to the default
        makebutton("Button5", gridbag, c); //another row
        c.gridwidth  =  GridBagConstraints.RELATIVE;  //next-to-last  in
row
        makebutton("Button6", gridbag, c);
        c.gridwidth = GridBagConstraints.REMAINDER; //end row
        makebutton("Button7", gridbag, c);
        c.gridwidth = 1;             //reset to the default
        c.gridheight = 2;
        c.weighty = 1.0;
        makebutton("Button8", gridbag, c);
        c.weighty = 0.0;             //reset to the default
        c.gridwidth = GridBagConstraints.REMAINDER; //end row
        c.gridheight = 1;            //reset to the default
        makebutton("Button9", gridbag, c);
        makebutton("Button10", gridbag, c);
setSize(300,300);        setTitle("DU");          setVisible(true);
           }}
```

```
import java.awt.*;
import java.awt.event.*;
public class Demo extends Frame {
protected void makebutton(String name,
                    GridBagLayout gridbag,
                    GridBagConstraints c)
{
      Button button = new Button(name);
      gridbag.setConstraints(button, c);
      add(button);
   }

public static void main(String args[]) {
             Demo obj=new Demo();
                    obj.showFrame();


  }
```

GridBagLayout

```
void showFrame(){
        GridBagLayout gridbag = new GridBagLayout();
        GridBagConstraints c = new GridBagConstraints();
        setFont(new Font("SansSerif", Font.PLAIN, 14));
        setLayout(gridbag);
        c.fill = GridBagConstraints.BOTH;
        c.weightx = 1.0;
        makebutton("Button1", gridbag, c);
        makebutton("Button2", gridbag, c);
        makebutton("Button3", gridbag, c);
        c.gridwidth = GridBagConstraints.REMAINDER; //end row
        makebutton("Button4", gridbag, c);
        c.weightx = 0.0;            //reset to the default
        makebutton("Button5", gridbag, c); //another row
        c.gridwidth  =  GridBagConstraints.RELATIVE;  //next-to-last  in
row
        makebutton("Button6", gridbag, c);
        c.gridwidth = GridBagConstraints.REMAINDER; //end row
        makebutton("Button7", gridbag, c);
        c.gridwidth = 1;            //reset to the default
        c.gridheight = 2;
        c.weighty = 1.0;
        makebutton("Button8", gridbag, c);
        c.weighty = 0.0;            //reset to the default
        c.gridwidth = GridBagConstraints.REMAINDER; //end row
        c.gridheight = 1;           //reset to the default
        makebutton("Button9", gridbag, c);
        makebutton("Button10", gridbag, c);
setSize(300,300);      setTitle("DU");       setVisible(true);
            }}
```

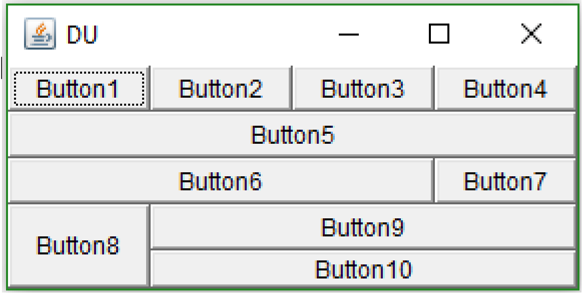
```
import java.awt.*;
import java.awt.event.*;
public class Demo extends Frame {
protected void makebutton(String name,
                    GridBagLayout gridbag,
                    GridBagConstraints c)
{
     Button button = new Button(name);
     gridbag.setConstraints(button, c);
     add(button);
   }

public static void main(String a
          Demo obj=new De
               obj.showF

  }
```



GridBagLayout

```
void showFrame(){
        GridBagLayout gridbag = new GridBagLayout();
        GridBagConstraints c = new GridBagConstraints();
        setFont(new Font("SansSerif", Font.PLAIN, 14));
        setLayout(gridbag);
        c.fill = GridBagConstraints.BOTH;
        c.weightx = 1.0;
        makebutton("Button1", gridbag, c);
        makebutton("Button2", gridbag, c);
        makebutton("Button3", gridbag, c);
        c.gridwidth = GridBagConstraints.REMAINDER; //end row
        makebutton("Button4", gridbag, c);
        c.weightx = 0.0;            //reset to the default
        makebutton("Button5", gridbag, c); //another row
        c.gridwidth  =  GridBagConstraints.RELATIVE;  //next-to-last  in
row
        makebutton("Button6", gridbag, c);
        c.gridwidth = GridBagConstraints.REMAINDER; //end row
        makebutton("Button7", gridbag, c);
        c.gridwidth = 1;            //reset to the default
        c.gridheight = 2;
        c.weighty = 1.0;
        makebutton("Button8", gridbag, c);
        c.weighty = 0.0;            //reset to the default
        c.gridwidth = GridBagConstraints.REMAINDER; //end row
        c.gridheight = 1;           //reset to the default
        makebutton("Button9", gridbag, c);
        makebutton("Button10", gridbag, c);
setSize(300,150);       setTitle("DU");         setVisible(true);
            }}
```

```
import java.awt.*;
import java.awt.event.*;
public class Demo extends Frame {
protected void makebutton(String name,
                        GridBagLayout gridbag,
                        GridBagConstraints c)
{
     Button button = new Button(name);
     gridbag.setConstraints(button, c);
     add(button);
   }

public static void main(String args[]) {
            Demo obj=new Demo();
                    obj.showF
  }
```



Dialog Container

- A Dialog is a top-level window with a title and a border that is typically used to take some form of input from the user.
- The default layout for a dialog is BorderLayout.
- A dialog may have another window as its owner when it's constructed.
- 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.
- Thus, other parts of your program remain active and accessible

Dialog Container

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

- Here, parentWindow is the owner of the dialog box. If mode is true, the dialog box is modal.
- Otherwise, it is modeless. The title of the dialog box can be passed in title. Generally, you will subclass Dialog, adding the functionality required by your application.

- Notice that when the dialog box is closed, we need to call dispose( ). This method is defined by Window, and it frees all system resources associated with the dialog box window.

Dialog Container

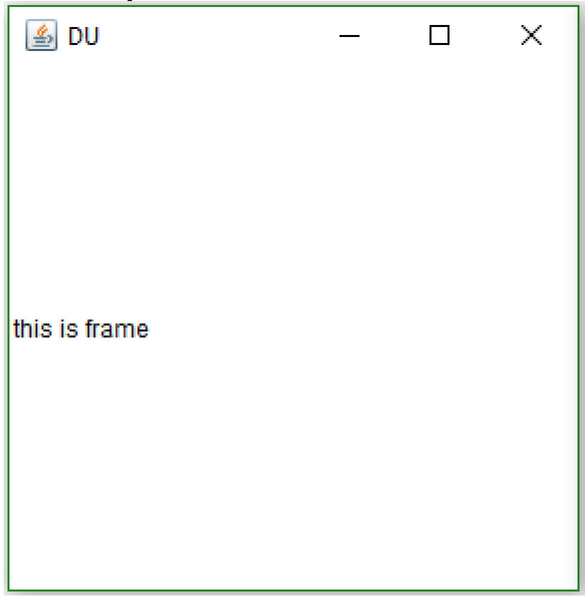
```
import java.awt.*;
import java.awt.event.*;
class Exam extends Frame implements ActionListener{
        static Dialog d;
        void showFrame(){
        add(new Label("this is frame"));
        setSize(300,300);
        setVisible(true);
        d = new Dialog(this , "Dialog Example", true);
        d.setLayout( new FlowLayout() );
        Button b = new Button ("OK");
        b.addActionListener (this);
        d.add( new Label ("Click button to continue."));
        d.add(b);
        d.setSize(300,300);
        d.setVisible(true);
```

```
public void actionPerformed( ActionEvent e )
{
            Exam.d.dispose();
}

  public static void main(String args[]) {
        Exam obj=new Exam();
                obj.showFrame();
      }
}
```

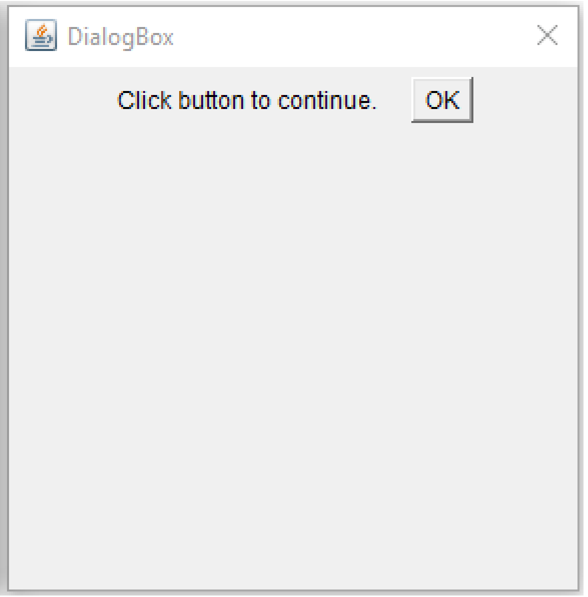
Dialog Container

```
import java.awt.*;
import java.awt.event.*;
class    Exam    extends    Frame    implements
ActionListener{
        static Dialog d;
        void showFrame(){
        add(new Label("this is frame"));
        setSize(300,300);
        setVisible(true);
        d = new Dialog(this , "Dialog Example", true);
        d.setLayout( new FlowLayout() );
        Button b = new Button ("OK");
        b.addActionListener (this);
        d.add(    new    Label    ("Click    button    to
continue."));
        d.add(b);
        d.setSize(300,300);
        d.setVisible(true);
```

```
public  void  actionPerformed(  ActionEvent  e  )
{
                Exam.d.dispose();
}

   public static void main(String args[]) {
        Exam obj=new Exam();
                obj.showFrame();
        }
}
```


DU

this is frame


DialogBox

Click button to continue. OK

Swing Containers

- javax.swing package provides classes for java swing components.
- Creation of swing containers and components is very similar to AWT based Containers and Components.
- Containers
    - ➢JPanel
    - ➢JFrame
    - ➢Jwindow
    - ➢JDialog

Swing components

- JButton
- JTextArea
- JTextField
- JCheckBox
- JLabel
- JList
- Jmenu
- Jpasswordfield
- JPanel
- JPopupMenu
- JProgressBar
- JScrollPane
- JComboBox
- JComponent
- JDialog
- JTextPane
- JToolBar
- JTree

Swings vs AWT

AWT	Swing
Platform dependent	Platform independent
Heavy weight	Light weight
Do not Support plug and feel	Support plug and feel
Less number of components	Large number of components