GUI Programming

In this Unit will do how you can *reuse* the graphics classes provided in JDK for constructing your own Graphical User Interface (GUI) applications.
There are current three sets of Java APIs for graphics programming:
AWT (Abstract Windowing Toolkit), Swing and JavaFX.

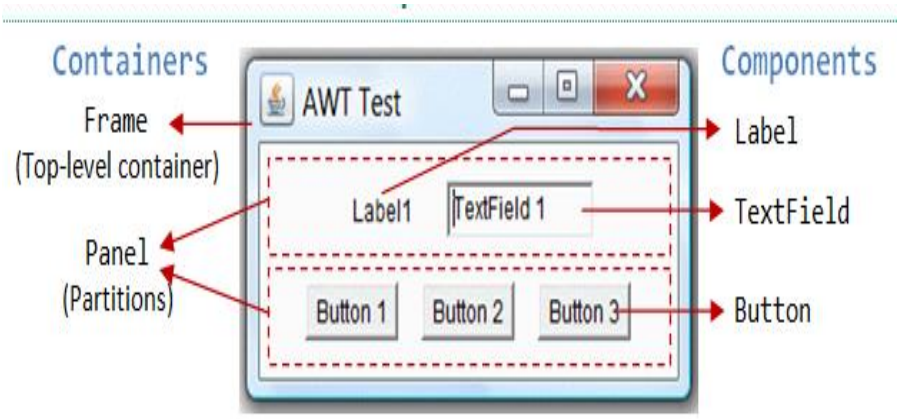
1. AWT API was introduced in JDK 1.0. Most of the AWT UI components have become obsolete and should be replaced by newer Swing UI components.
2. Swing API, a much more comprehensive set of graphics libraries that enhances the AWT, was introduced as part of Java Foundation Classes (JFC) after the release of JDK 1.1. JFC consists of Swing, Java2D, Accessibility, Internationalization, and Pluggable Look-and-Feel Support APIs. JFC has been integrated into core Java since JDK 1.2.
3. The latest JavaFX, which was integrated into JDK 8, was meant to replace Swing. JavaFX was moved out from the JDK in JDK 11, but still available as a separate module.

Programming GUI with AWT

I shall start with the AWT before moving into Swing to give you a complete picture of Java Graphics.
AWT is huge! It consists of 12 packages of 370 classes (Swing is even bigger, with 18 packages of 737 classes as of JDK 8).
Fortunately, only 2 packages - java.awt and java.awt.event - are commonly-used.

1. The java.awt package contains the *core* AWT graphics classes:
   - GUI Component classes, such as Button, TextField, and Label.
   - GUI Container classes, such as Frame and Panel.
   - Layout managers, such as FlowLayout, BorderLayout and GridLayout.
   - Custom graphics classes, such as Graphics, Color and Font.
2. The java.awt.event package supports event handling:
   - Event classes, such as ActionEvent, MouseEvent, KeyEvent and WindowEvent,
   - Event Listener Interfaces, such as ActionListener, MouseListener, MouseMotionListener, KeyListener and WindowListener,
   - Event Listener Adapter classes, such as MouseAdapter, KeyAdapter, and WindowAdapter.

AWT provides a **platform-*independent* and *device-independent*** interface to develop graphic programs that runs on all platforms, including Windows, macOS, and UNIXs.



There are two groups of GUI elements:

1. *Component* : Components are elementary GUI entities, such as Button, Label, and TextField.

2. *Container*: Containers, such as Frame and Panel, are used to *hold components in a specific layout* (such as FlowLayout or GridLayout). A container can also hold sub-containers.

In the above figure, there are three containers: a Frame and two Panels.

A Frame is the *top-level container* of an AWT program. A Frame has a title bar (containing an icon, a title, and the minimize/maximize/close buttons), an optional menu bar and the content display area.

A Panel is a *rectangular area* used to group related GUI components in a certain layout.

In the above figure, the top-level Frame contains two Panels. There are five components: a Label (providing description), a TextField (for users to enter text), and three Buttons (for user to trigger certain programmed actions).

In a GUI program, a component must be kept (or added) in a container. You need to identify a container to hold the components.

AWT Container Classes

There are four types of containers in Java AWT:

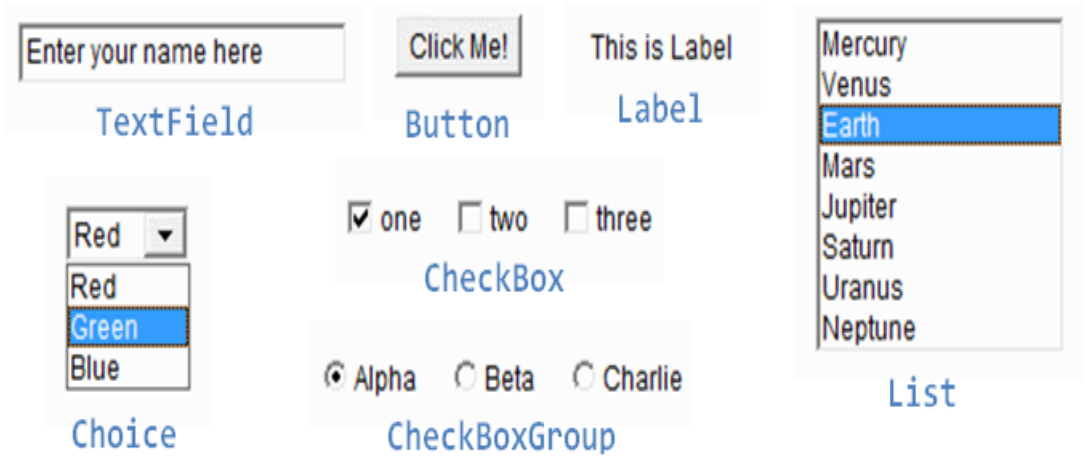
Window: Window is a top-level container that represents a graphical window or dialog box. The Window class extends the Container class, which means it can contain other components, such as buttons, labels, and text fields.

Panel: Panel is a container class in Java. It is a lightweight container that can be used for grouping other components together within a window or a frame.

Frame: The Frame is the container that contains the title bar and border and can have menu bars.

Dialog: A dialog box is a temporary window an application creates to retrieve user input.

AWT Component Classes



Every container has a method called add(Component c).
A container (say aContainer) can invoke

aContainer.add(aComponent) to add aComponent into itself.

For example,

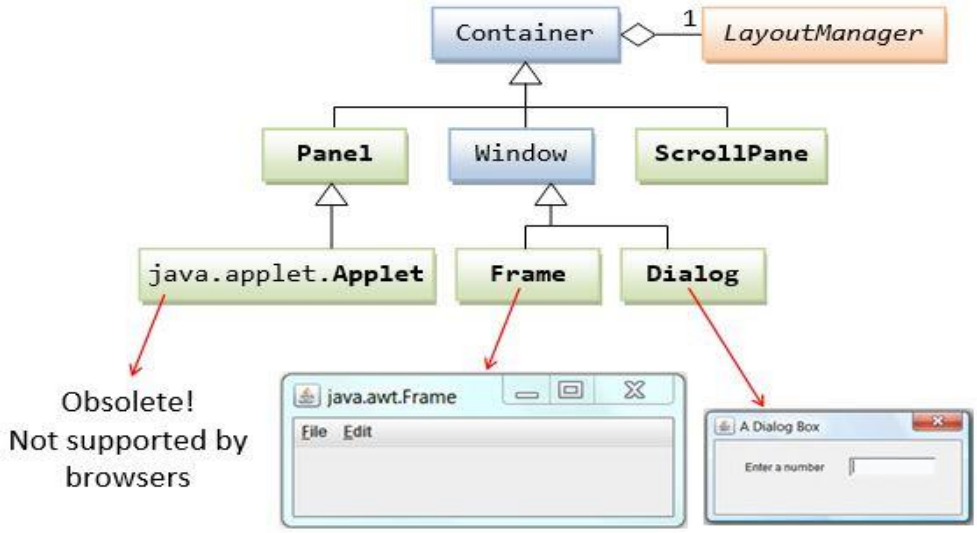
```
Panel pnl = new Panel();          // Panel is a container
Button btn = new Button("Press"); // Button is a component
pnl.add(btn);                     // The Panel container adds a Button
```

Swing

Java has extended the AWT with the Swing Set, which consists of lightweight components that can be drawn directly onto containers using code written in Java. The event classes, the event listener interfaces, and others java Swing GUI, we need at least one container object.

There are 3 types of Java Swing containers.

1. **Panel**: It is a pure container and is not a window in itself. The sole purpose of a Panel is to organize the components on to a window.
2. **Frame**: It is a fully functioning window with its title and icons.
3. **Dialog**: It can be thought of like a pop-up window that pops out when a message has to be displayed. It is not a fully functioning window like the Frame.

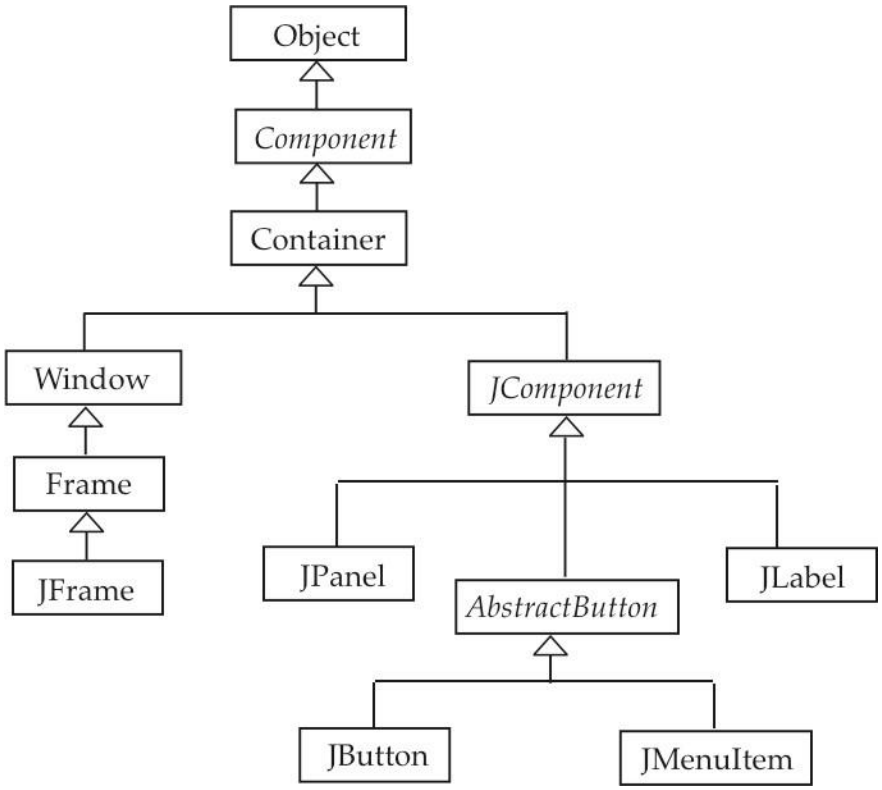


Comparison of AWT and Swing

java.awt	**javax.swing**
Frame	JFrame
Panel	JPanel
Canvas	JPanel
Label	JLabel
Button	JButton
TextField	JTextField
Checkbox	JCheckBox
List	JList

Choice JComboBox

Part of Class Hierarchy



LayOut Managers

GridLayout

GridLayout divides a surface into rows and columns forming cellsthat may each hold one component.

Execute this method in the frame Constructor:

getContentPane().setLayout(**new** GridLayout(2, 4));

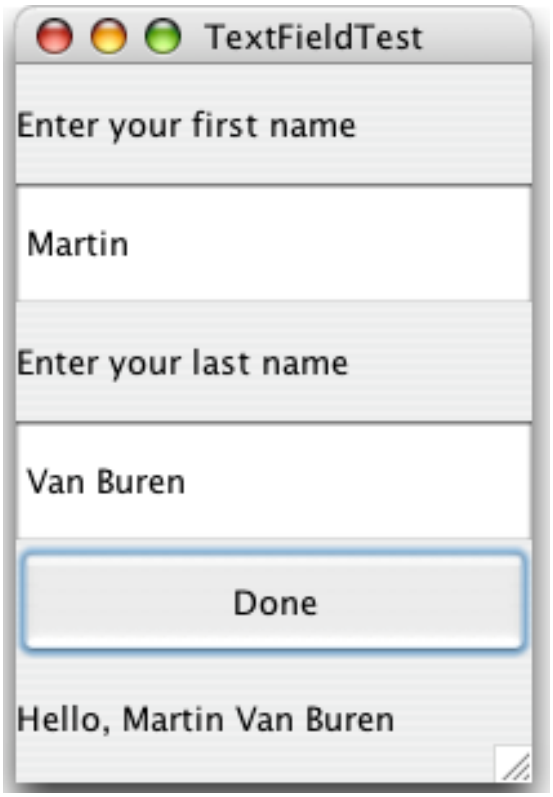
Component are added to the layout using the instance method add(component), which places the items left-to-right in the rows,top-to-bottom.

1	2	3	4
5	6	7	8

Components placed in a grid region take the size of the region.

Example: TextFieldTest cp.setLayout(**new**

GridLayout(6,1));



Flow Layout

The Java FlowLayout class is used to arrange the components in a line, one after another (in a flow). It is the default layout of the applet or panel.

Fields of FlowLayout class

public static final int LEFT
public static final int RIGHT
public static final int CENTER
public static final int LEADING
public static final int TRAILING

Constructors of FlowLayout class

FlowLayout(): creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
FlowLayout(int align): creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
FlowLayout(int align, int hgap, int vgap): creates a flow layout with the given alignment and the given horizontal and vertical gap.
Example of FlowLayout class: Using FlowLayout() constructor

FileName: FlowLayoutExample.java

```
// import statements
import java.awt.*;
import javax.swing.*;

public class FlowLayoutExample
{

JFrame frameObj;

// constructor
FlowLayoutExample()
{
   // creating a frame object
   frameObj = new JFrame();
```

```
    // creating the buttons
    JButton b1 = new JButton("1");
    JButton b2 = new JButton("2");
    JButton b3 = new JButton("3");
    JButton b4 = new JButton("4");
    JButton b5 = new JButton("5");
    JButton b6 = new JButton("6");
    JButton b7 = new JButton("7");
    JButton b8 = new JButton("8");
    JButton b9 = new JButton("9");
    JButton b10 = new JButton("10");


    // adding the buttons to frame
    frameObj.add(b1); frameObj.add(b2); frameObj.add(b3); frameObj.add(b4);
    frameObj.add(b5); frameObj.add(b6);  frameObj.add(b7);  frameObj.add(b8);
    frameObj.add(b9);  frameObj.add(b10);

    // parameter less constructor is used
    // therefore, alignment is center
    // horizontal as well as the vertical gap is 5 units.
    frameObj.setLayout(new FlowLayout());

    frameObj.setSize(300, 300);
    frameObj.setVisible(true);
}

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

Border Layout

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 a frame or window. The BorderLayout provides five constants for each region:

public static final int NORTH
public static final int SOUTH
public static final int EAST
public static final int WEST
public static final int CENTER

Constructors of BorderLayout class:

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

FileName: Border.java

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

public class Border
{
JFrame f;
Border()
{
   f = new JFrame();

    // creating buttons
   JButton b1 = new JButton("NORTH");; // the button will be labeled as NORTH
   JButton b2 = new JButton("SOUTH");; // the button will be labeled as SOUTH
   JButton b3 = new JButton("EAST");; // the button will be labeled as EAST
   JButton b4 = new JButton("WEST");; // the button will be labeled as WEST
   JButton b5 = new JButton("CENTER");; // the button will be labeled as CENTER
```

```
    f.add(b1, BorderLayout.NORTH); // b1 will be placed in the North Direction
    f.add(b2, BorderLayout.SOUTH);  // b2 will be placed in the South Direction
    f.add(b3, BorderLayout.EAST);  // b2 will be placed in the East Direction
    f.add(b4, BorderLayout.WEST);  // b2 will be placed in the West Direction
    f.add(b5, BorderLayout.CENTER);  // b2 will be placed in the Center

    f.setSize(300, 300);
    f.setVisible(true);
}
public static void main(String[] args) {
    new Border();
}
}
```

Java CardLayout

The Java 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.

Constructors of CardLayout Class

CardLayout(): creates a card layout with zero horizontal and vertical gap.
CardLayout(int hgap, int vgap): creates a card layout with the given horizontal and vertical gap.

Commonly Used Methods of CardLayout Class

public void next(Container parent): is used to flip to the next card of the given container.
public void previous(Container parent): is used to flip to the previous card of the given container.
public void first(Container parent): is used to flip to the first card of the given container.
public void last(Container parent): is used to flip to the last card of the given container.
public void show(Container parent, String name): is used to flip to the specified card with the given name.

FileName: CardLayoutExample1.java

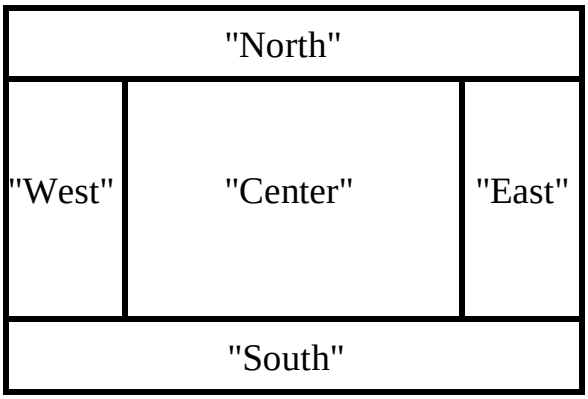
```
// import statements


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

public class CardLayoutExample1 extends JFrame implements ActionListener
{

CardLayout crd;

// button variables to hold the references of buttons
JButton btn1, btn2, btn3;
Container cPane;

// constructor of the class
CardLayoutExample1()
{

cPane = getContentPane();

//default constructor used
// therefore, components will
// cover the whole area
crd = new CardLayout();

cPane.setLayout(crd);

// creating the buttons
btn1 = new JButton("Apple");
btn2 = new JButton("Boy");
btn3 = new JButton("Cat");

// adding listeners to it
btn1.addActionListener(this);
btn2.addActionListener(this);
btn3.addActionListener(this);
```

```
cPane.add("a", btn1); // first card is the button btn1
cPane.add("b", btn2); // first card is the button btn2
cPane.add("c", btn3);  // first card is the button btn3

}
public void actionPerformed(ActionEvent e)
{
// Upon clicking the button, the next card of the container is shown
// after the last card, again, the first card of the container is shown upon clicking
crd.next(cPane);
}

// main method
public static void main(String argvs[])
{
// creating an object of the class CardLayoutExample1
CardLayoutExample1 crdl = new CardLayoutExample1();

// size is 300 * 300
crdl.setSize(300, 300);
crdl.setVisible(true);
crdl.setDefaultCloseOperation(EXIT_ON_CLOSE);
}
}
```

JFrame

A window with a title bar, a resizable border, and possiblya menu bar.

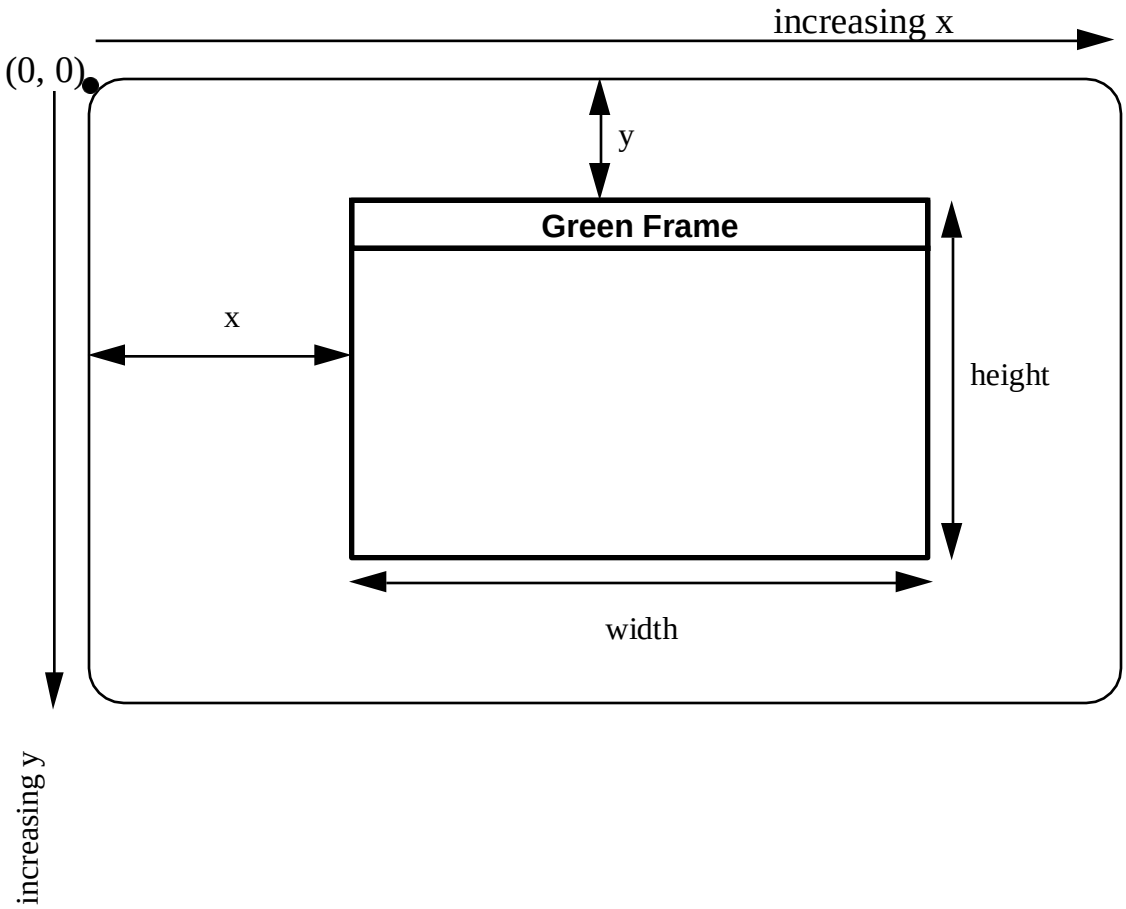
- A frame is not attached to any other surface.
- It has a content pane that acts as a container.
- The container uses BorderLayout by default.



- A layout manager, an instance of one of the layout classes,describes how components are placed on a container.

Creating a JFrame

```
JFrame jf = new JFrame("title");          // or JFrame() jf.setSize(300,
200);                                     // width, height in pixels (required)
jf.setVisible(true);                      // (required)
jf.setTitle("New Title");
jf.setLocation(50, 100);                  // x and y from upper-left corner
```



Placing Components

Container cp = jf.getContentPane();cp.add(c1, "North");
cp.add(c2, "South");

or cp.add(c1, BorderLayout.NORTH);

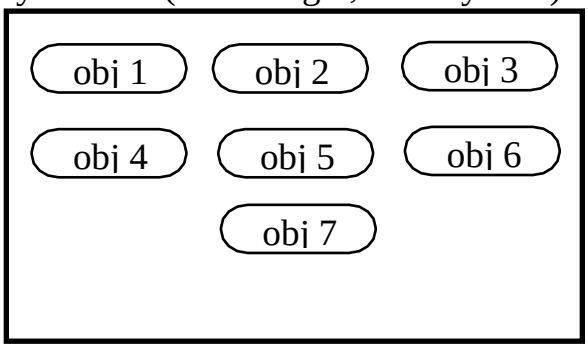
// Java has five constants like this

The constant BorderLayout.NORTH has the value "North".

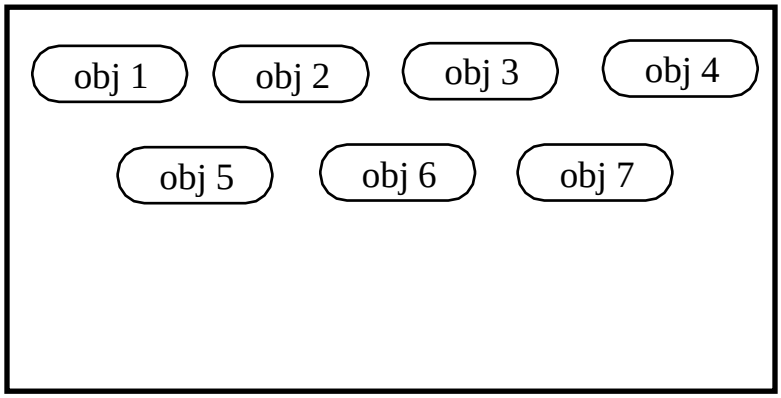
JPanel

An invisible Container used to hold components or to draw on.

- Uses FlowLayout by default (left-to-right, row-by-row).



If the container is resized, the componentswill adjust themselves to new positions.



- Any component may be placed on a panel using

  add.JPanel jp = **new** JPanel();

  jp.add(componentObj);

- A panel has a Graphics object that controls its surface.

  Subclass JPanel and override the method

  void paintComponent(Graphics g)to describe the surface of the panel.

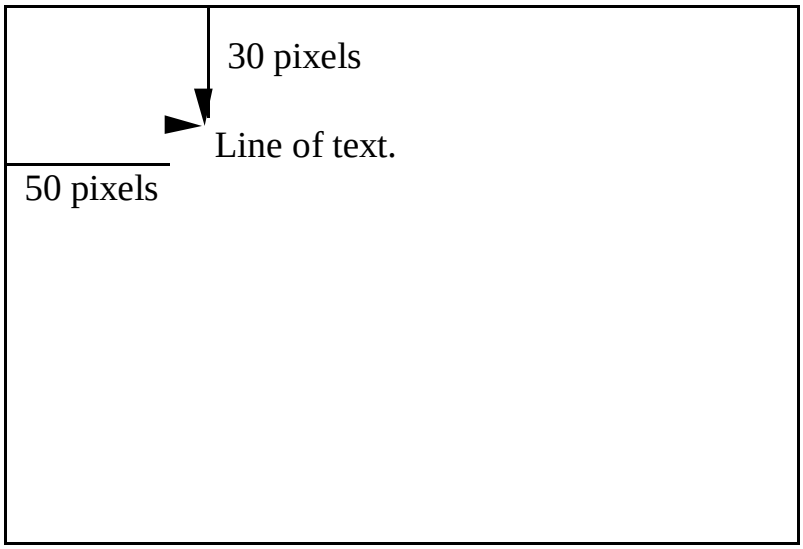
Default: Blank background matching existing background color.

Set the background to a particular color using: jp.setBackground(**new** Color(255, 204,153));

Drawing Tools for Graphics Object g

g.drawString("Line of text.", 50, 30)

Draws the string with its left baseline at (50, 30) usingthe current Font.



g.drawRect(100, 30, 100, 150)

Draws a rectangle whose upper left corner is at (100,30)on the panel and whose width and height are 100 and 150, respectively.

g.drawRoundRect(15, 40, 25, 15, 5, 5);

Draws a solid rectangle whose upper left point is (15,40), whose width is 25, and whose height is 15 and where the diameter of the corner circles is 5.

g.fillOval(100, 100, 20, 30);

Draws a solid oval whose upper left point is (100,100),whose width is 20, and whose height is 30.

- Positions on a panel are specified in pixels measured fromthe upper left corner, horizontal pixels first and vertical pixels second.

Useful Methods (in java.awt.Graphics)

void drawRect(**int** x, **int** y, **int** width, **int** height);

void drawRoundRect(**int** x, **int** y, **int** w, **int** h,
int arcWidth, **int** arcHeight);

void drawOval(**int** x, **int** y, **int** width, **int** height);**void**

fillRect(**int** x, **int** y, **int** width, **int** height); **void**

fillRoundRect(**int** x, **int** y, **int** w, **int** h,
int arcWidth, **int** arcHeight);

void fillOval(**int** x, **int** y, **int** width, **int** height);**void**

drawString(String text, **int** x, **int** y); **void** drawLine(**int**

x1, **int** y1, **int** x2, **int** y2);

void draw3DRect(**int** x, **int** y, **int** w, **int** h, **boolean** raised);

void fill3DRect(**int** x, **int** y, **int** w, **int** h, **boolean** raised);

void drawArc(**int** x, **int** y, **int** w, **int** h,**int** startAngle, **int** arcAngle);
void fillArc(**int** x, **int** y, **int** w, **int** h,**int** startAngle, **int** arcAngle);

void setColor(Color c);

void setFont(Font f);

void drawPolygon(**int** [] x, **int** [] y, **int** numPoints);

void fillPolygon(**int** [] x, **int** [] y, **int** numPoints);

Components Of SWING class

JLabel

Labels are components consisting of a String to be placed on acontainer.

- Used to label another component.
- Used as output on a container.

```
JLabel lab = new JLabel("Enter name: ");
lab.setText("New  Message");
String s = lab.getText();
```

JButton

Buttons are components that can be placed on a container.

- Can have a String label on them.

```
JButton b =
    new  JButton("Click");
    add(b);                    // assuming FlowLayout
    b.setLabel("Press");
    String s = b.getLabel();
```

- Register an ActionListener using

```
b.addActionListener(new  ButtonHandler());
```

- An event is generated when the mouse clicks on a Button.
- The implementation of *actionPerformed* in ButtonHandlerdescribes the response to the button press.

Since the classes that implement listeners normally need access to identifiers in the class that defined the graphical interface, they are defined as inner classes frequently.

The parameter to *actionPerformed* is an ActionEvent, a classdefined in the package *java.awt.event*.

Example

Prompt user to press a button and then display resultas a Label.

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

public class ButtonTest extends JFrame
{
    private JLabel result;

    ButtonTest()
    {
        setTitle("ButtonTest");

        JLabel lab = new JLabel("Click one of the buttons.");getContentPane().add(lab,
        "North");

        result = new JLabel(" ");
        getContentPane().add(result,  "South");

        JButton yes = new JButton("Yes"); getContentPane().add(yes,
        "West");
        yes.addActionListener(new   YesHandler());

        JButton no = new JButton("No"); getContentPane().add(no, "East");
        no.addActionListener(new NoHandler());
    }
```

```
    class YesHandler implements ActionListener
    {
        public void actionPerformed(ActionEvent evt)
        {   result.setText("You pressed the Yes button.");                    }
    }

    class NoHandler implements ActionListener
    {
        public void actionPerformed(ActionEvent evt)
        {   result.setText("You pressed the No button."); }
    }


    public static void main(String [] args)
    {   JFrame jf = new ButtonTest();
        jf.setSize(250, 150);
        jf.setVisible(true);
    }
}
```

Note that clicking the close box of the window does notterminate the program, although in Swing the window disappears.

The frame generates a window closing event, but we haveregistered no listener for this event.

We need to write code that recognizes the event fired when thewindow is closed and shuts down the program.

JTextField

Text fields allow the user to enter text that can be processedby the program.

One constructor takes an **int** representing the width of the fieldin characters (approximately). Others take an initial string as a parameter or both.

```
JTextField tf1 = new JTextField(10); JTextField tf2 =
new JTextField("Message");add(tf1);
add(tf2);
tf1.setText("New Message");String s =
tf2.getText();
tf2.setEditable(false);                    // default is true
```

Entering return inside a text field triggers an ActionEvent, whichis sent to all registered ActionListeners.

In implementations of actionPerformed(), the String in the JTextField may be fetched (input) or a new String can be placed inthe field (output).

Example

Allow user to enter first and last names in text fields and sayhello to the person named.
Let the main class be the ActionListener.

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

public class TextFieldTest extends JFrame implements ActionListener
{
    TextFieldTest()
    {
```

```
        setTitle("TextFieldTest"); addWindowListener(new
        WindowHandler());
        Container cp = getContentPane();
        cp.setLayout(new  FlowLayout());                  // override default
        cp.add(new JLabel("Enter your first name"));
        cp.add(first = new JTextField(15));
        cp.add(new JLabel("Enter your last name"));cp.add(last = new
        JTextField(15));
        JButton done = new JButton("Done");
        cp.add(done); done.addActionListener(this);
        result = new JLabel("**********");cp.add(result);
}

public void actionPerformed(ActionEvent e)
{
        String firstName = first.getText();String
        lastName = last.getText();
        result.setText("Hello, " + firstName + " " + lastName); }
}

class WindowHandler extends WindowAdapter
{
        public void windowClosing(WindowEvent e)
        { System.exit(0);  }
}


private JLabel result;

private JTextField first, last;


public static void main(String [] a)
{
        JFrame jf = new TextFieldTest();
```

```
        jf.setSize(160, 200); jf.setVisible(true);
    }
}
```

JComboBox (drop-down list)

A list of items from which a user can select only one item at atime.

When a selection is made, an ActionEvent is fired.

```
JComboBox scope = new JComboBox();
scope.addItem("private");
scope.addItem("protected");
scope.addItem("default"); s
scope.addItem("public");

getContentPane().add(scope,  "South");

scope.addActionListener(new  ComboHandler());

class ComboHandler implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        JComboBox jcb = (JComboBox)e.getSource();
        String value = (String)jcb.getSelectedItem();
            // compare value with "private", "protected", etc.
```

Alternative Construction

```
Object [] items = {“private", "protected", "default", "public”};

JComboBox scope = new JComboBox(items);
```

JList (“menu” of choices)

This component contains list of items from which a user canselect one or several items at a time.
Any number of items can be made visible in the list.
When an item is selected or deselected, a ListSelectionEventis fired.

It calls method *valueChanged* in the interfaceListSelectionListener.

These classes and interfaces are found in the package *javax.swing.event*.

A JList can be constructed in several ways.

One constructor takes an array of objects as its parameter.

```
String [] list = { "Java", "C", "Pascal", "Ada", "ML", "Prolog" };

JList lang = new JList(list);
```

A JList does not come with a scrollbar automatically.

We need to place it onto a scroll pane by passing it to a JScrollPane constructor.

```
JScrollPane jsp = new JScrollPane(lang);

getContentPane().add(jsp, "West");
```

Methods

```
lang.setVisibleRowCount(2);
lang.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
lang.addListSelectionListener(new ListHandler());
```

Selection Choices ListSelectionModel.SINGLE_SELECTION
ListSelectionModel.SINGLE_INTERVAL_SELECTION
ListSelectionModel.MULTIPLE_INTERVAL_SELECTION (default)

Selections

Click:	single selection Control-
click:	additional selection
Shift-click:	contiguous range

JTextArea

A text area allows multiple lines of text. It can be used for both input and output.

Constructors

JTextArea(**int** rows, **int** cols)

JTextArea(String text, **int** rows, **int** cols)

```
JTextArea ta = new JTextArea("Message", 5, 20);
ta.setEditable(false);                    // default is true
ta.select(3,7);              // character positions 3, 4, 5, and 6
ta.selectAll();
String s = ta.getSelectedText();ta.setText("Replace
all text"); ta.append("A line of text\n");
```

Problem: Print strings from a String array *sArr* into a text area.

```
JTextArea lines = new JTextArea("", 10, 60);

for (int k=0; k< sArr.length; k++)
      lines.append(sArr[k] + "\n");
```

A Password Field

Suppose we need a text field for entering a password.
As characters are typed in the field, we do not want them to be visible.
Solution: JPasswordField, a subclass of JTextField.
Constructors
JPasswordField()
JPasswordField(String s)
Copyright 2004 by Ken Slonneger GUI Programming 55
JPasswordField(int w)
JPasswordField(String s, int w)
Example

```
JPasswordField jpf =
new JPasswordField("Enter your password here");
```

The echo character, which shows as the user types, is an asterisk (*) by default.
The echo character can be changed using:
jpf.setEchoChar('?');
If the enter (return) character is typed in the field, an ActionEvent is fired.
The contents of the field are provided by the method:
jpf.getPassword(), which returns an array of char.
Sample Code
This program has a password field on a frame.
When return is typed in the field, it compares the string typed with a predefined password, “herky”.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class JPApp extends JFrame
{
private String pw = "herky";
private JPasswordField pwField;
private JLabel ans;
```



```
JPApp()
{
setTitle("Password Verification");
Font font = new Font("SanSerif", Font.PLAIN, 18);
Container cp = getContentPane();
JPanel panel = new JPanel();
JLabel jl = new JLabel("Enter Password: ");
jl.setFont(font);
panel.add(jl);
pwField = new JPasswordField(12);
pwField.setFont(font);
pwField.setEchoChar('#');
panel.add(pwField);
cp.add(panel, BorderLayout.CENTER);
ans = new JLabel();
ans.setFont(font);
cp.add(ans, BorderLayout.SOUTH);
pwField.addActionListener(new ActionListener()
{
```

```
public void actionPerformed(ActionEvent e)
{
String password =
new String(pwField.getPassword());
if (pw.equals(password))
ans.setText("Access Granted");
else
ans.setText("Access Denied");
}
} );
}
```

```
void getFocus()
{
pwField.requestFocus();
}
public static void main(String [] args)
{
JPApp jpa = new JPApp();
jpa.addWindowListener(new WindowAdapter()
{
public void windowClosing(WindowEvent e)
{ System.exit(0); }
} );
jpa.setSize(500,200);
jpa.setVisible(true);
jpa.getFocus(); // give field the focus
}
}
```