GUI Programming

Based on the Abstract Windowing Toolkit, AWT, found in the package java.awt.

- AWT components
- called heavyweight components
- implemented with native native code (probably C++) written for the particular computer.

Disadvantages:

- Not uniform between platforms.
- Not very fast.
- Not very flexible.

The AWT library was written in six weeks.

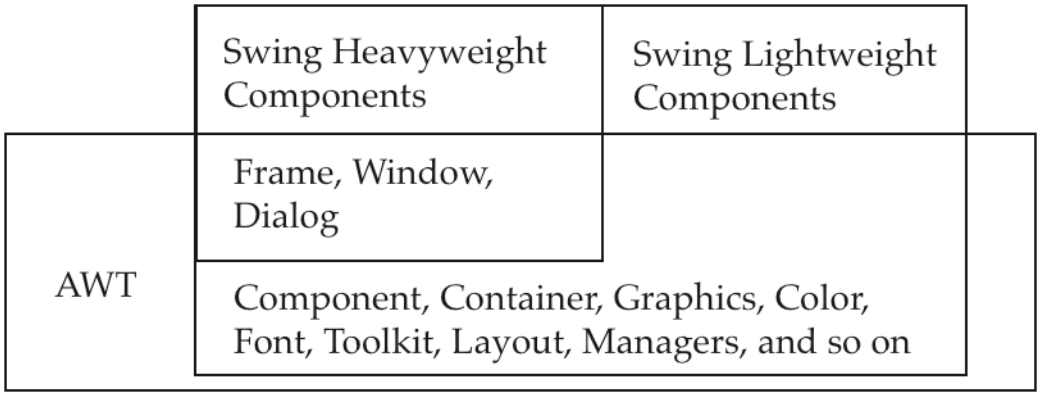
AWT Classes

Although new versions of most of the components have been provided, many of the classes and interfaces in the AWT are still used for GUI programming.

Component, Container, Color, Font, FontMetrics, EventObject, the event classes, the event listener interfaces, and others.

Swing

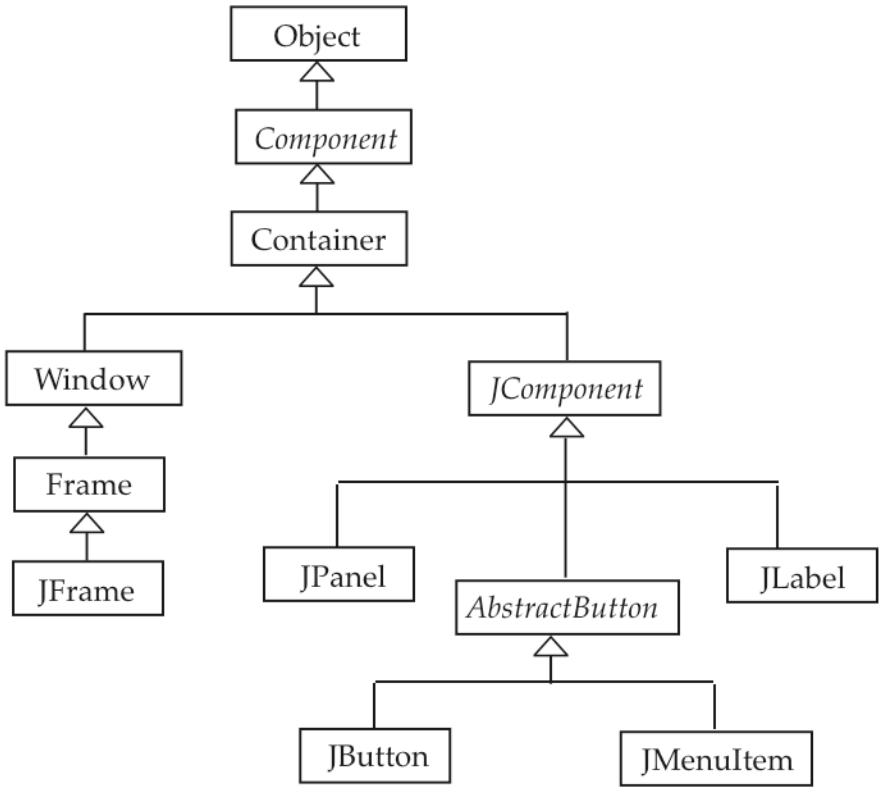
Version 1.2 of Java has extended the AWT with the Swing Set, which consists of lightweight components that can be drawn directly onto containers using code written in Java.



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

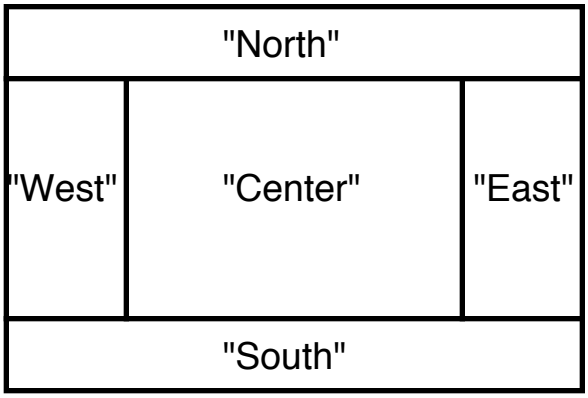
Part of Class Hierarchy



JFrame

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

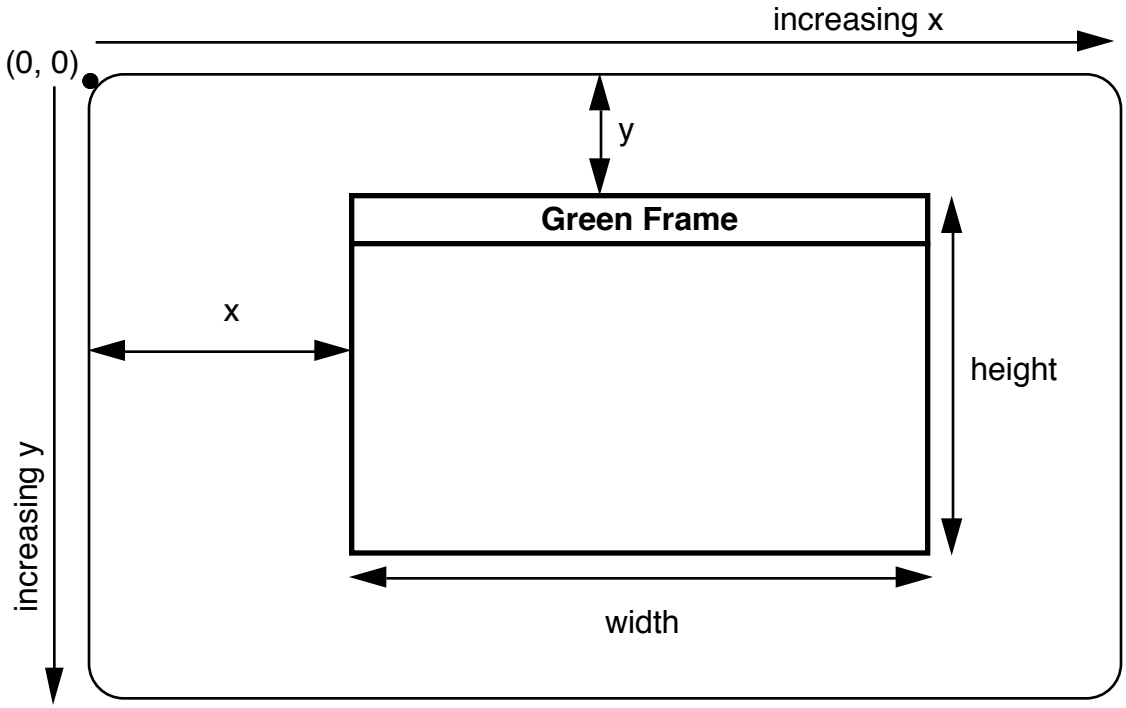
- A frame is not attached to any other surface.
- It has a content pane that acts as a container.
- The container uses BorderLayout by default.



- 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);`

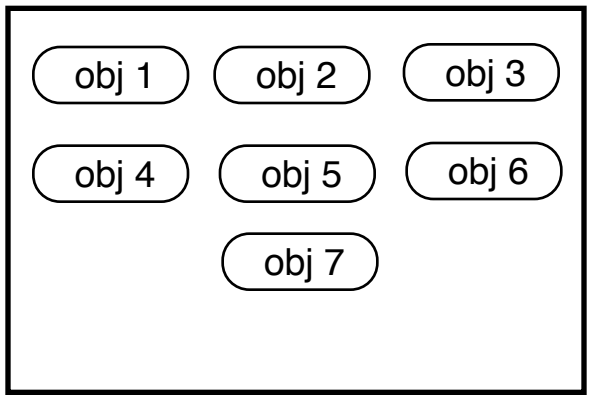
```
// Java has five constants like this
```

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

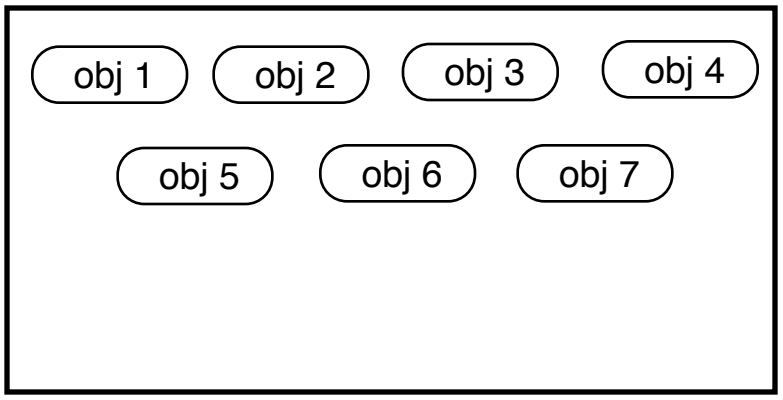
JPanel

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 components will 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.

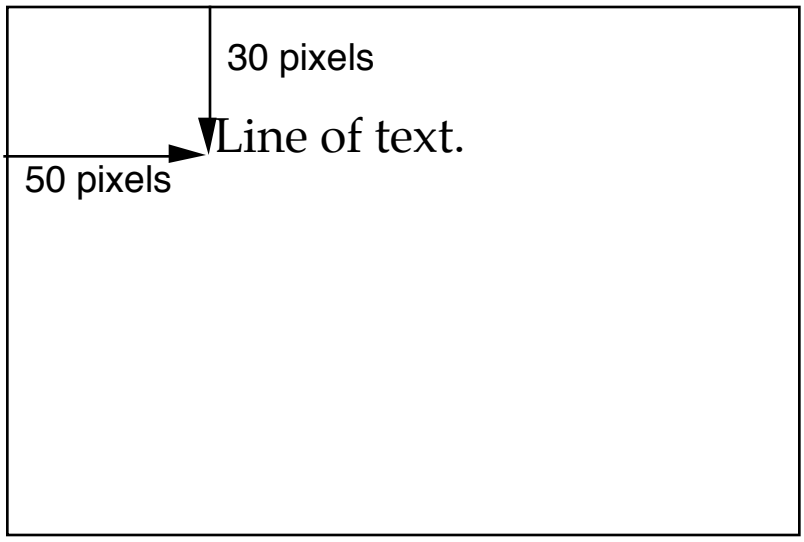
Set the background to a particular color using:

```
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) using the 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 from the 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);
```

Color Class

Color objects can be created using a constructor:

public Color(**int** red, **int** green, **int** blue)

where each **int** value satisfies 0≤val≤255.

For example:

Color mine = **new** Color(255, 204, 153);

Color provides 13 constants

	Red	Green	Blue
Color.black	0	0	0
Color.darkGray	64	64	64
Color.gray	128	128	128
Color.lightGray	192	192	192
Color.white	255	255	255
Color.red	255	0	0
Color.green	0	255	0
Color.blue	0	0	255
Color.cyan	0	255	255
Color.magenta	255	0	255
Color.yellow	255	255	0
Color.orange	255	200	0
Color.pink	255	175	175

Font Class

Font objects can be created using the constructor:

public Font(String name, **int** style, **int** points)

and the constants: Font.PLAIN, Font.BOLD, and Font.ITALIC.

Examples

```
Font f1 = new Font("Helvetica", Font.BOLD, 18);
Font f2 = new Font("Courier", Font.PLAIN, 12);
```

Logical Names (preferred)

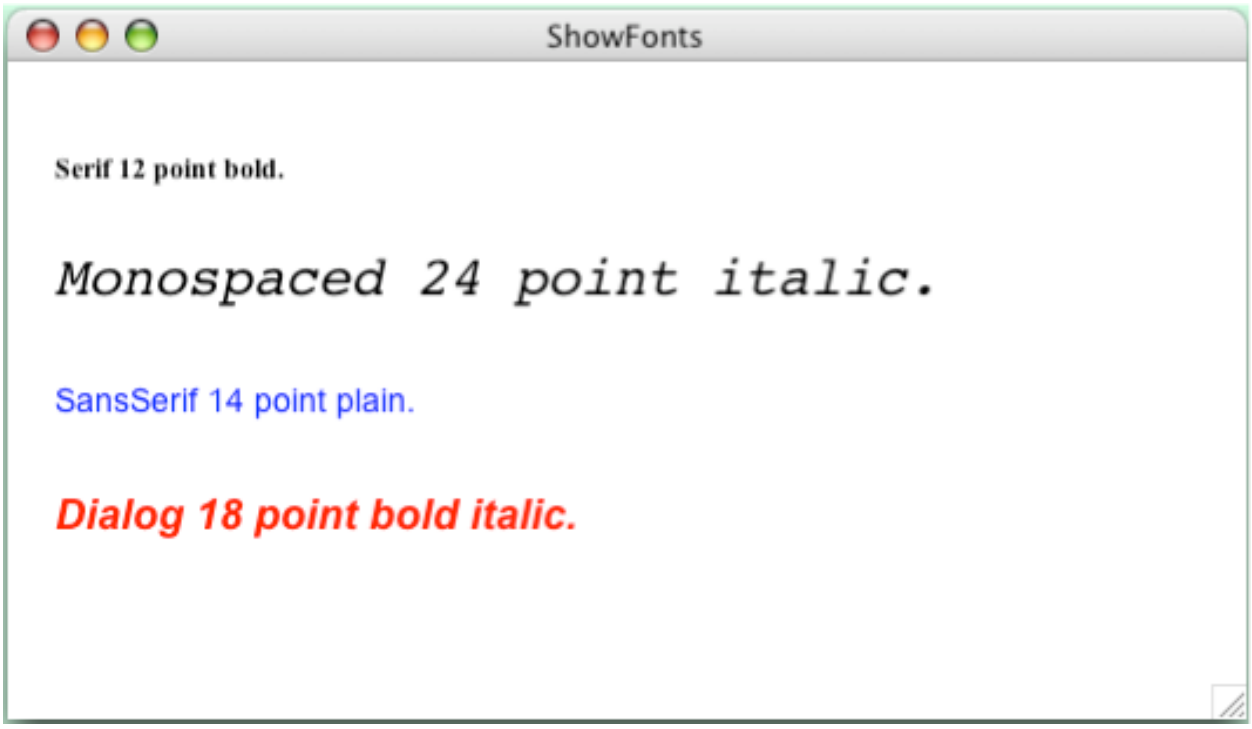
Dialog, DialogInput, Monospaced, Serif, SansSerif.

Example

Write four lines of text on a panel that is placed on a frame. The lines illustrate different fonts.

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

class FontPanel extends JPanel
{
   FontPanel()
   {  setBackground(Color.white);
   }

   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);          // clear background
      g.setFont(new Font("Serif", Font.BOLD, 12));
      g.drawString("Serif 12 point bold.", 20, 50 );
      g.setFont(new Font("Monospaced", Font.ITALIC, 24));
      g.drawString("Monospaced 24 point italic.", 20, 100);
```

```
        g.setColor(Color.blue);
        g.setFont(new Font("SansSerif", Font.PLAIN, 14));
        g.drawString("SansSerif 14 point plain.", 20, 150);
        g.setColor(Color.red);
        g.setFont(new Font("Dialog", Font.BOLD + Font.ITALIC, 18));
        g.drawString(g.getFont().getName() + " " +
              g.getFont().getSize() + " point bold italic.", 20, 200);
    }
}

class Fonts extends JFrame
{
    Fonts()
    {  setTitle("ShowFonts");
        setSize(500, 400);
        setLocation(200, 200);
        Container contentPane = getContentPane();
        contentPane.add(new FontPanel());     // default = "Center"
    }
}

public class ShowFonts
{
    public static void main(String [] args)
    {
        JFrame jf = new Fonts();
        jf.setVisible(true);
    }
}
```



JLabel

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

- 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();          // rarely used
```

Handling Events (Delegation model)

Events originate from component objects such as buttons, menus, windows, and so on.

Each event source maintains a (possibly empty) list of listeners, which are objects that expect to be notified when an event occurs.

The programmer registers listener objects with the appropriate components.

A listener is an object whose class implements a listener interface for the corresponding event source.

When an event occurs, it is sent to all registered listener objects, each of which implements the methods of the interface designed for that event.

Action events come from the components:

JButton	JTextField
JMenuItem	JRadioButton
JMenu	JCheckBox

When one of these components produces an action event, the component notifies the ActionListener's registered with it.

An ActionListener interface expects the method

public void actionPerformed(ActionEvent e)

to be implemented, and this method is called when the event is dispatched.

The behavior that we want performed in response to the action is coded in the *actionPerformed* method.

Use the method

```
com.addActionListener(ActionListener aL)
```

to register a listener object with the component com.

Next we study some of the components that issue events.

Later we summarize the listener interfaces and their methods.

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 ButtonHandler describes 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 class defined in the package *java.awt.event*.

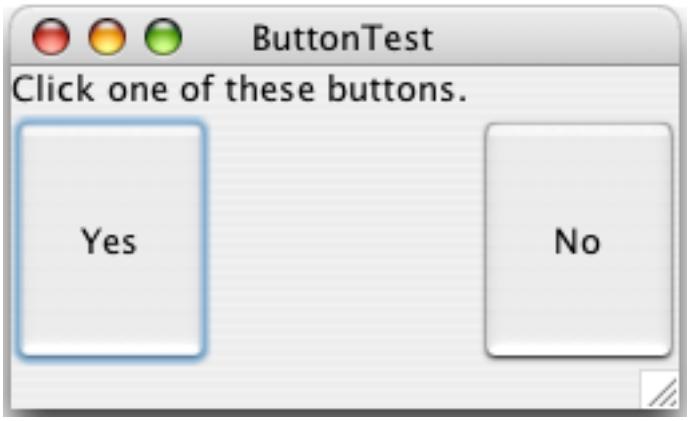
Example

Prompt user to press a button and then display result as 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 not terminate the program, although in Swing the window disappears.

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

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

Closing the Window

We must handle the window closing event to make the program terminate.

We implement the interface WindowListener, which contains a method

```
windowClosing(WindowEvent we)
```

for defining the appropriate behavior.

In fact, the WindowListener interface has *seven* methods, all of which must be implemented.

```
public void windowClosing(WindowEvent e)
public void windowClosed(WindowEvent e)
public void windowIconified(WindowEvent e)
public void windowOpened(WindowEvent e)
public void windowDeiconified(WindowEvent e)
public void windowActivated(WindowEvent e)
public void windowDeactivated(WindowEvent e)
```

ButtonTest needs two changes to enable the program to respond to window closing events.

In the Constructor

```
ButtonTest()
{
        setTitle("ButtonTest");
        addWindowListener(new WindowHandler());
                :
}
```

Another Inner Class

```
class WindowHandler implements WindowListener
{
   public void windowClosing(WindowEvent e)
   {  System.exit(0);  }
   public void windowClosed(WindowEvent e)  { }
   public void windowIconified(WindowEvent e)  { }
   public void windowOpened(WindowEvent e)  { }
   public void windowDeiconified(WindowEvent e)  { }
   public void windowActivated(WindowEvent e)  { }
   public void windowDeactivated(WindowEvent e)  { }
}
```

Adapters

Observe the large number of methods that need to be implemented in the WindowHandler, even though only one of the methods does anything.

As a convenience, Java provides classes, called adapter classes, that implement the listener interfaces with multiple methods using method definitions that do nothing.

We can avoid mentioning these do-nothing methods by subclassing the adapter classes, which are found in the package *java.awt.event*.

Using the adapter for windows, the inner class in ButtonTest can be written:

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

Since the class WindowAdapter implements the interface WindowListener, the new class WindowHandler does also.

Anonymous Inner Class

The anonymous inner class mechanism is ideal for this situtation.

- Want to define a subclass of WindowAdapter.
- Body of subclass is small.
- Subclass is used only once.

Combine two pieces of code on the previous pages into one:

```
addWindowListener(new WindowAdapter()
{
      public void windowClosing(WindowEvent e)
      {  System.exit(0);  }
} );
```


The anonymous inner class

```
WindowAdapter()
{     public void windowClosing(WindowEvent e)
      {  System.exit(0);  }
}
```

is registered as a Window Listener although it is given no name.

Listener Interfaces

The table below shows the number of methods in some of the interfaces and the name of the corresponding adapter, if there is one.

Listener	Methods	Adapter
ActionListener	1	
AdjustmentListener	1	
ItemListener	1	
KeyListener	3	KeyAdapter
MouseListener	5	MouseAdapter
MouseMotionListener	2	MouseMotionAdapter
WindowListener	7	WindowAdapter
ListSelectionListener	1	
ChangeListener	1	

Listener Interface Methods

java.awt.event

ActionListener

    actionPerformed(ActionEvent e)

ItemListener

    itemStateChanged(ItemEvent e)

MouseListener

    mousePressed(MouseEvent e)
    mouseReleased(MouseEvent e)
    mouseEntered(MouseEvent e)
    mouseExited(MouseEvent e)
    mouseClicked(MouseEvent e)

MouseMotionListener

    mouseDragged(MouseEvent e)
    mouseMoved(MouseEvent e)

KeyListener

    keyPressed(KeyEvent e)
    keyReleased(KeyEvent e)
    keyTyped(KeyEvent e)

WindowListener

windowClosing(WindowEvent e)
windowOpened(WindowEvent e)
windowIconified(WindowEvent e)
windowDeiconified(WindowEvent e)
windowClosed(WindowEvent e)
windowActivated(WindowEvent e)
windowDeactivated(WindowEvent e)

AdjustmentListener

adjustmentValueChanged(AdjustmentEvent e)

javax.swing.event

ChangeListener

stateChanged(ChangeEvent e)

ListSelectionListener

valueChanged(ListSelectionEvent e)

Events, Event Sources and their Interfaces

ActionEvent	JButton JMenu JMenuItem JRadioButton JCheckBox JTextField	ActionListener
ItemEvent	JButton JMenu JMenuItem JRadioButton JCheckBox	ItemListener
KeyEvent	Component	KeyListener
MouseEvent	Component	MouseListener MouseMotionListener
AdjustmentEvent	JScrollBar	AdjustmentListener
WindowEvent	JFrame JDialog	WindowListener
ListSelectionEvent	JList	ListSelectionListener
ChangeEvent	JSlider	ChangeListener

JTextField

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

One constructor takes an **int** representing the width of the field in 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, which is 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 in the field (output).

Example

Allow user to enter first and last names in text fields and say hello 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);
    }
}
```

GridLayout

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

Execute this method in the frame Constructor:

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

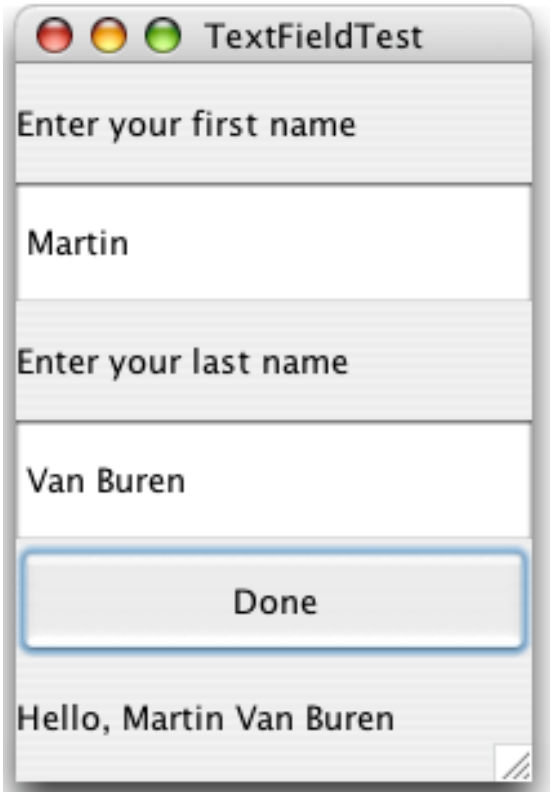
Component are added to the layout using the instance method add(component), which places the items left-to-right in the rows, top-to-bottom.

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));
```



Combining Listeners

Create a frame with three buttons labeled Yellow, Cyan, and Magenta. When one is pressed, the background changes to that color.

Use constants in the class Color.

Buttons will be a subclass of JFrame that provides the three buttons for indicating the background color.

An inner class BH implements ActionListener.

The BH class must therefore contain a definition of the method *actionPerformed*.

Two versions of *actionPerformed* are shown in the example, both of which determine which button was pressed.

The window closing event is handled by an anonymous inner class that extends WindowAdapter.

Code for Buttons

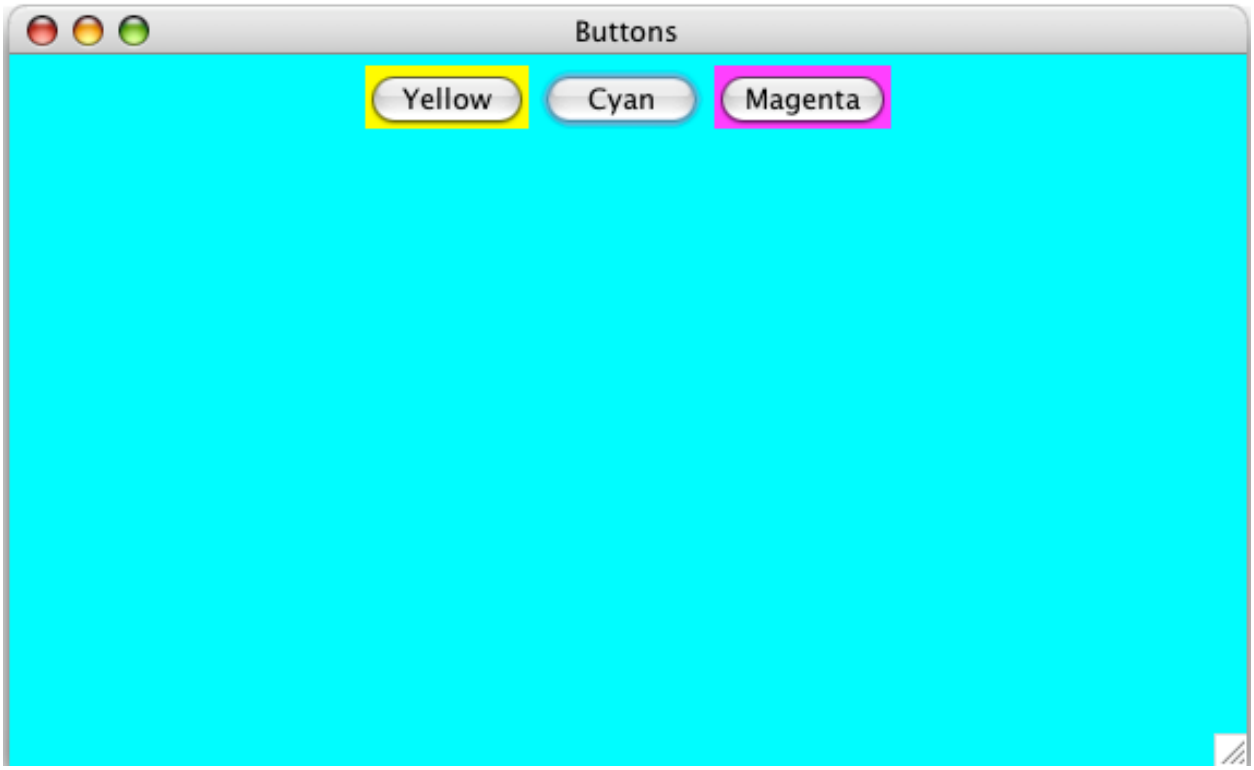
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Buttons extends JFrame
{
    private JButton yellow, cyan, magenta;
    private JPanel jp;
```

```
Buttons()
{
    setTitle("Buttons");
    addWindowListener( new WindowAdapter()
    {  public void windowClosing(WindowEvent e)
       {  System.exit(0);  }
    } );

    jp = new JPanel();

    BH bh = new BH();

    yellow = new JButton("Yellow");
    yellow.setBackground(Color.yellow);
    jp.add(yellow);
    yellow.addActionListener(bh);

    cyan = new JButton("Cyan");
    cyan.setBackground(Color.cyan);
    jp.add(cyan);
    cyan.addActionListener(bh);

    magenta = new JButton("Magenta");
    magenta.setBackground(Color.magenta);
    jp.add(magenta);
    magenta.addActionListener(bh);

    getContentPane().add(jp);
}
```



```
class BH implements ActionListener
{
// Instance method getSource for an EventObject returns
// a reference to the component that caused the event.
    public void actionPerformed(ActionEvent evt)
    {
        Object ob = evt.getSource();
        if (ob==yellow)
                        jp.setBackground(Color.yellow);
        else if (ob==cyan)
                        jp.setBackground(Color.cyan);
        else if (ob==magenta)
                        jp.setBackground(Color.magenta);
        jp.repaint();
    }

// To use ob as a JButton, it must be downcasted, say
//      ((JButton)ob).getText().


/*****************************************************************
// Instance method getActionCommand for an ActionEvent
// returns the String label of component that caused the event.
    public void actionPerformed(ActionEvent evt)
    {
        String arg = evt.getActionCommand();

        if (arg.equals("Yellow"))
                        jp.setBackground(Color.yellow);

        else if (arg.equals("Cyan"))
                        jp.setBackground(Color.cyan);
```

```
            else if (arg.equals("Magenta"))
                        jp.setBackground(Color.magenta);

            jp.repaint();
        }

    ************************************************************/
    }    // end of inner class

    public static void main(String [] args)
    {
        JFrame jf = new Buttons();
        jf.setSize(300,200);
        jf.setVisible(true);
    }
}
```



Customized Listeners

Avoid decision making in the button handler by customizing the listeners for the particular buttons.

```
class BH implements ActionListener
{
    private Color color;

    BH(Color c)
    { color = c; }

    public void actionPerformed(ActionEvent evt)
    {
        jp.setBackground(color);
        jp.repaint();
    }
}
```

When handlers are registered with their buttons, pass the appropriate color for the button.

```
yellow.addActionListener(new BH(Color.yellow));
                :
cyan.addActionListener(new BH(Color.cyan));
                :
magenta .addActionListener(new BH(Color.magenta));
                :
```

The rest of the code remains the same.

JCheckBox

A check box has one of two states: checked or unchecked.

These constructors takes a String that sits next to the check box as a label and an optional boolean value.

```
JCheckBox happy = new JCheckBox("Happy", true);
JCheckBox hungry = new JCheckBox("Hungry");

add(happy);
add(hungry);
```

A change in a check box triggers both an ActionEvent and an ItemEvent, which can be recognized by an ActionListener or an ItemListener that has been registered using:

```
happy.addActionListener(new ActHandler());
happy.addItemListener(new ItemHandler());
```

The state of a check box can be ascertained using the method

```
happy.isSelected()
```

which returns a boolean value.

Example

Place two check boxes on a frame and record their state in a label message.

As a shorthand, we use the frame class as the ItemListener.

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



```
public class CheckBoxTest extends JFrame
                          implements ItemListener
{
   CheckBoxTest()
   {
      setTitle("Check Box Test");
      addWindowListener(new WindowAdapter()
      {  public void windowClosing(WindowEvent e)
         {  System.exit(0);  }
      } );

      Container cp = getContentPane();
      cp.setBackground(Color.white);
      Font font = new Font("SansSerif", Font.PLAIN, 18));
      JLabel lab =
            new JLabel("Click the checkboxes:", JLabel.CENTER);
      lab.setFont(font);
      cp.add(lab, BorderLayout.NORTH);
      result =
            new JLabel("You checked nothing.", JLabel.CENTER);
      result.setFont(font);
      cp.add(result, BorderLayout.SOUTH);

      yes = new JCheckBox("Yes");
      yes.setFont(font);
      yes.setBackground(Color.white);
      cp.add(yes, BorderLayout.WEST);
      yes.addItemListener(this);
      no = new JCheckBox ("No");
      no.setFont(font);
      no.setBackground(Color.white);
      cp.add(no, BorderLayout.EAST);
      no.addItemListener(this);
   }
```

```
    public void itemStateChanged(ItemEvent evt)
    {
        if (yes.isSelected() && no.isSelected())
            result.setText("You checked both boxes.");
        else if (yes.isSelected())
            result.setText("You checked the Yes box.");
        else if (no.isSelected())
            result.setText("You checked the No box.");
        else
            result.setText("You checked neither box.");
    }

    private JLabel result;
    private JCheckBox yes, no;

    public static void main(String [] args)
    {   JFrame jf = new CheckBoxTest();
        jf.setSize(600, 400);
        jf.setVisible(true);
    }
}
```

Alternative: Use ActionListener

```
public class CheckBoxTest extends JFrame
                                  implements ActionListener
    yes.addActionListener(this);
    no.addActionListener(this);


  public void actionPerformed(ActionEvent evt)
  {
    if (yes.isSelected() && no.isSelected ())
          result.setText("You checked both boxes.");
     :
```

Why not have the CheckBoxTest class implement WindowListener as well ItemListener or ActionListener?

Radio Buttons

Similar to check boxes, except that only one button in a group may be checked at a time.

```
hot = new JRadioButton("Hot", false);
warm = new JRadioButton("Warm", true);
cold = new JRadioButton("Cold", false);

ButtonGroup gp = new ButtonGroup();
gp.add(hot);
gp.add(warm);
gp.add(cold);
```

The box for *warm* is the selected one, initially.

Add radio buttons to a container jp:

```
jp.add(hot);    jp.add(warm);        jp.add(cold);
```

Any change in a radio button triggers an ActionEvent and an ItemEvent.

Use the method *getSource* on the ActionEvent or the ItemEvent to determine which radio button fired the event (which was just selected).

Menus

Menus are placed on a menu bar, which is attached to a frame.

```
JMenuBar mb = new JMenuBar();

JMenu file = new JMenu("File");
file.add(new JMenuItem("New"));          // no way to add
file.add(new JMenuItem("Open"));         // listeners
mb.add(file);

JMenu edit = new JMenu("Edit");
edit.add(copy = new JMenuItem("Copy"));
edit.add(paste = new JMenuItem("Paste"));
edit.addSeparator();
edit.add(clear = new JMenuItem("Clear"));
mb.add(edit);

jf.setJMenuBar(mb);              // assumes jf is a JFrame
```

Menu Events

Selecting a menu item triggers an ActionEvent that can be handled by an ActionListener that must be register with each menu item that it plans to cover.

If *ae* is the ActionEvent sent by some menu item, it can be decoded using the following commands:

```
Object jmi = ae.getSource();
if (jmi == clear)  handleClear();
```

or

```
String arg = ((JMenuItem)jmi).getText();
```



```
if (arg.equals("Clear"))  handleClear();
```

Note that *getSource* returns an Object, which may need to be downcasted.

Example

Put three radio buttons at the top of a frame that changes color according to the button chosen.

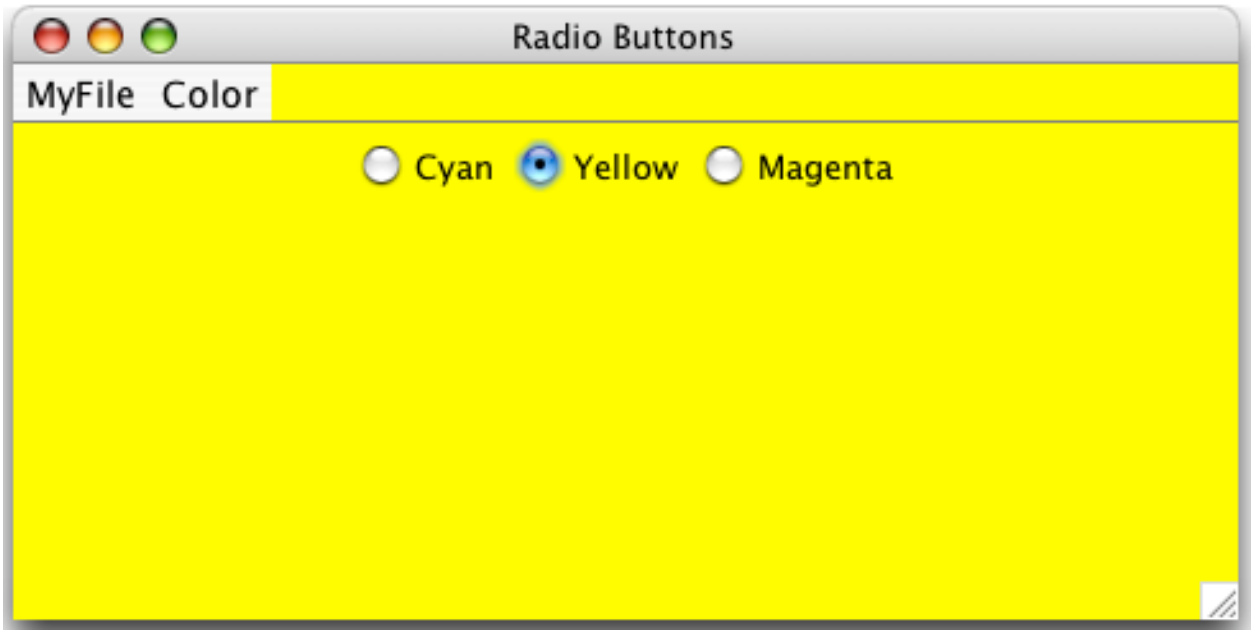
Also provide a menu that allows the color to be picked. Have the buttons denote the selected color for consistency.

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

class RadioButtonFrame extends JFrame
                              implements ActionListener
{
   RadioButtonFrame()
   {
      setTitle("Radio Buttons");

      addWindowListener(new WindowAdapter()
      {
         public void windowClosing(WindowEvent e)
         {  System.exit(0);
         }
      } );

      topPanel = new JPanel();

      ButtonGroup g = new ButtonGroup();
      cyanButton = addRadioButton(topPanel, g, "Cyan", false);
      yellowButton = addRadioButton(topPanel, g, "Yellow", false);
      magentaButton =
               addRadioButton(topPanel, g, "Magenta", false);
```

```
    getContentPane().add(topPanel, "North");
    panel = new JPanel();
    getContentPane().add(panel, "Center");
    JMenuBar mb = new JMenuBar();
    JMenu file = new JMenu("MyFile");
    JMenuItem quit = new JMenuItem("Quit");
    file.add(quit);
    quit.addActionListener(this);
    mb.add(file);
    JMenu color = new JMenu("Color");
    JMenuItem cyanItem = new JMenuItem("Cyan");
    color.add(cyanItem);
    cyanItem.addActionListener(this);
    JMenuItem yellowItem = new JMenuItem("Yellow");
    color.add(yellowItem);
    yellowItem.addActionListener(this);
    JMenuItem magentaItem = new JMenuItem("Magenta");
    color.add(magentaItem);
    magentaItem.addActionListener(this);
    mb.add(color);
    setJMenuBar(mb);
  }
  JRadioButton addRadioButton(JPanel jp,
                      ButtonGroup g, String name, boolean v)
  {
    JRadioButton b = new JRadioButton(name, v);
    b.addActionListener(this);
    g.add(b);
    jp.add(b);
```

```
    return b;
  }

  public void actionPerformed(ActionEvent e)
  {
    Object source = e.getSource();
    if (source == cyanButton)
      setColor(Color.cyan);
    else if (source == yellowButton)
      setColor(Color.yellow);
    else if (source == magentaButton)
      setColor(Color.magenta);
    else
    {
      JMenuItem selection = (JMenuItem)source;
      String cmd = selection.getText();

      if (cmd.equals("Quit"))
        System.exit(0);
      else if (cmd.equals("Cyan"))
      {
        setColor(Color.cyan);
        cyanButton.setSelected(true);
      }
      else if (cmd.equals("Yellow"))
      {
        setColor(Color.yellow);
        yellowButton.setSelected(true);
      }
      else if (cmd.equals("Magenta"))
      {
        setColor(Color.magenta);
        magentaButton.setSelected(true);
      }
    }
  }
```

```
    private void setColor(Color c)
    {
        panel.setBackground(c);
        panel.repaint();
        topPanel.setBackground(c);
        topPanel.repaint();
    }
    private JPanel panel, topPanel;
    private JRadioButton cyanButton, yellowButton, magentaButton;
}
public class RadioButtons
{
    public static void main(String[] args)
    {
        JFrame jf = new RadioButtonFrame();
        jf.setSize(500, 300);
        jf.setVisible(true);
    }
}
```



JComboBox (drop-down list)

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

When a selection is made, an ActionEvent is fired.

```
JComboBox scope = new JComboBox();
scope.addItem("private");
scope.addItem("protected");
scope.addItem("default");
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 can select 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 ListSelectionEvent is fired.

It calls method *valueChanged* in the interface ListSelectionListener.

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

An Event Handler

```
class ListHandler implements ListSelectionListener
{
    public void valueChanged(ListSelectionEvent e)
    {
        JList jlist = (JList)e.getSource();
        Object [] values = jlist.getSelectedValues();
        // The string objects in the array may have to be
        // cast as String objects one at a time to be used.
```

Illegal Cast (throws ClassCastException)

```
String [] sa = (String [])values;
```

Rules for Downcasting

```
A a = get an object;
B b = (B)a;
```

1. B is a subclass of A.
2. *a* refers to an object of class B.

String [] is a subclass of Object [],
but *values* refers to an Object [], not a String [].

Example

Use a JComboBox to set the color of a group of words chosen from a JList object and displayed on a panel.

Draw the words on the panel via Graphics object associated with it.

Provide an ActionListener for the JComboBox and a ListSelectionListener for the JList.

Both the JList and JComboBox are placed in the East region using a vertical BoxLayout, which lets them have their natural height.

The words are drawn on a subclass Show of JPanel, which is placed in the center of the frame.

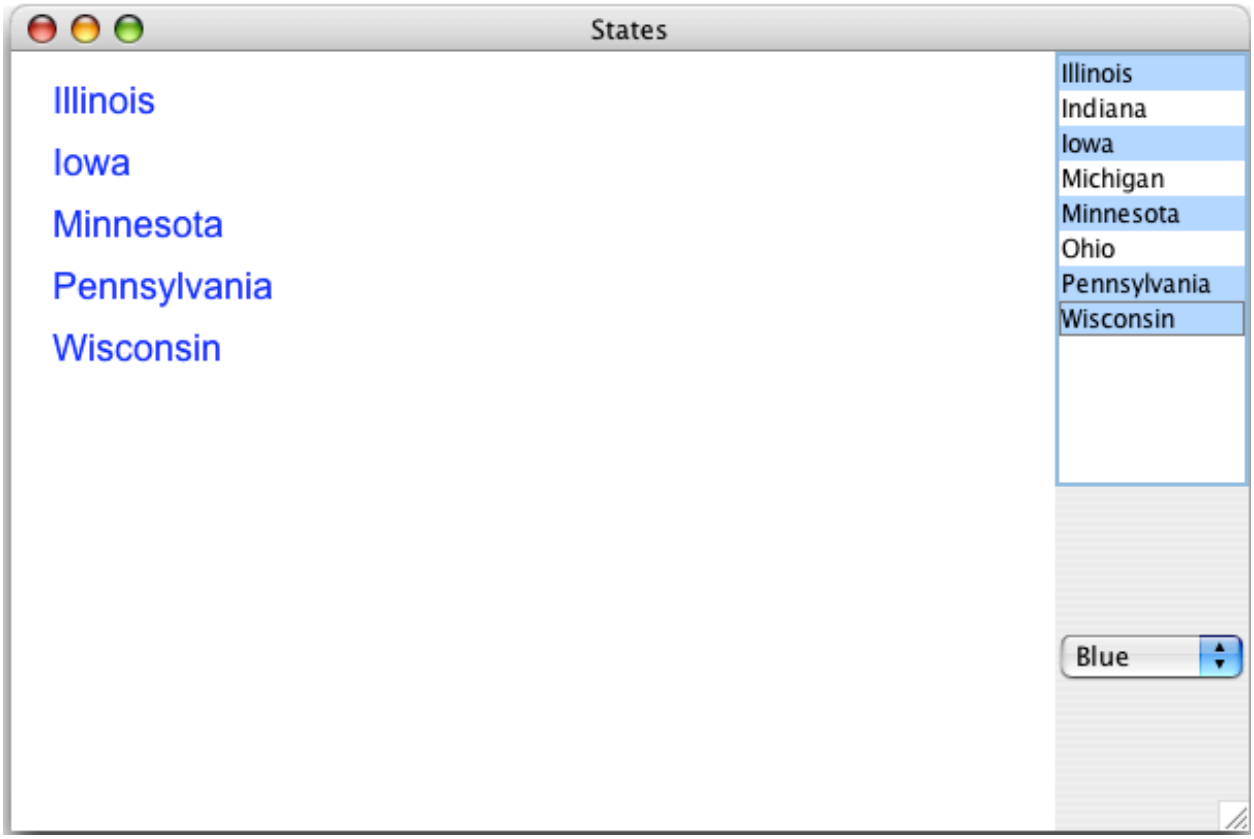
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class States extends JFrame
                implements ActionListener, ListSelectionListener
{
   States()
   {
      setTitle("States");
      setSize(600, 400);

      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      JComboBox choice = new JComboBox();
      choice.addItem("Red");        choice.addItem("Orange");
      choice.addItem("Green");      choice.addItem("Black");
      choice.addItem("Blue");       choice.addItem("Cyan");

      choice.addActionListener(this);

      String [] items = { "Illinois", "Indiana", "Iowa", "Michigan",
               "Minnesota", "Ohio", "Pennsylvania", "Wisconsin" };
      JList words = new JList(items);
      JScrollPane sPane = new JScrollPane(words);
      words.addListSelectionListener(this);

      JPanel jp = new JPanel();
      jp.setLayout(new BoxLayout(jp, BoxLayout.Y_AXIS));
      jp.add(sPane);              jp.add(choice);
      getContentPane().add(jp, "East");

      show = new Show();
      show.setBackground(Color.white);
      getContentPane().add(show, "Center");
   }
```



```
   private Color color = Color.red;
   private String [] states = { };       // initialized if color chosen first
   private Show show;

   public void actionPerformed(ActionEvent e)
   {
      JComboBox source = (JComboBox)e.getSource();
      Object item = source.getSelectedItem();
      if (item.equals("Red"))  color = Color.red;
      else if (item.equals("Green"))  color = Color.green;
      else if (item.equals("Blue"))  color = Color.blue;
      else if (item.equals("Orange"))  color = Color.orange;
      else if (item.equals("Black"))  color = Color.black;
      else if (item.equals("Cyan"))  color = Color.cyan;
      show.updatePanel(color, states);
   }

   public void valueChanged(ListSelectionEvent e)
   {
      JList source = (JList)e.getSource();
      Object [] values = source.getSelectedValues();
      states = new String [values.length];
      for (int k=0; k<values.length; k++)
         states[k] = (String)values[k];
      show.updatePanel(color, states);
   }

   public static void main(String [] args)
   {
      JFrame jf = new States();
      jf.setVisible(true);
   }
}
```

The Show panel stores its own copies of the color and the states list (an array of String) and overrides *paintComponent* to draw the words on the surface. Positions of *drawString* computed so that words are drawn neatly.

```
class Show extends JPanel
{
   private Color color;
   private String [] list = { };      // initialized for first paint

   void updatePanel(Color c, String [] s)
   {
      color = c;
      list = s;
      repaint();
   }

   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);

      Font font = new Font("SansSerif", Font.PLAIN, 18);
      g.setFont(font);
      g.setColor(color);

      for (int k=0; k<list.length; k++)
            g.drawString(list[k], 20, 30 +30*k);
   }
}
```

Using Anonymous Classes

Change the following lines in the class States and remove the two listener methods.

```
public class States extends JFrame
{
      :
      choice.addActionListener(new ActionListener()
      {
         public void actionPerformed(ActionEvent e)
         {
            JComboBox source = (JComboBox)e.getSource();
            Object item = source.getSelectedItem();
            if (item.equals("Red"))  color = Color.red;
            else if (item.equals("Green"))  color = Color.green;
            else if (item.equals("Blue"))  color = Color.blue;
            else if (item.equals("Orange"))  color = Color.orange;
```

```
            else if (item.equals("Black"))  color = Color.black;
            else if (item.equals("Cyan"))  color = Color.cyan;
            show.updatePanel(color, states);
        }
    } );
    :
    words.addListSelectionListener(new ListSelectionListener()
    {
        public void valueChanged(ListSelectionEvent e)
        {
            JList source = (JList)e.getSource();
            Object [] values = source.getSelectedValues();
            states = new String [values.length];
            for (int k=0; k<values.length; k++)
                states[k] = (String)values[k];
            show.updatePanel(color, states);
        }
    } );
```

Dynamic JLists

To have a JList change during the execution of a program, we must deal with its underlying model, which contains the actual data.

Example

- Create a model for a list

  ```
  DefaultListModel model = new DefaultListModel();
  ```

- Create the list

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

- Add items to the list by adding them to the model

  ```
  model.addElement("one");
  model.addElement("two");
  model.add(1, "oneandhalf");
  ```


- Remove items from the list by removing them from the model
  model.remove(0);
  model.removeElement("two");
- Clear the list using
  model.clear()
- Update list to agree with a new model
  list.setModel(model)
- Size of list returned by
  model.getSize()

Drawing on a Panel

Write a Java application that allows the user to draw on a panel.

- Application creates a panel on a frame for drawing.
- Uses Point class to save points.

Variations of mouseDown

- Click
- Shift-click
- Double-click

Uses of mouseDrag

- Draw a line (draw)
- Draw a filled oval (erase)

Point Class

An object of the Point class encapsulates two integer values, representing the x and y coordinates of a point.

Observe that the values are stored in public instance variables, a design decision that has been criticized.

```
public class java.awt.Point extends java.lang.Object
{
// Fields
        public int x;
        public int y;

// Constructors
        public Point();                 // (0, 0)
        public Point(int x, int y);
        public Point(Point p);

// Methods
        public boolean equals(Object obj);
        public void move(int nx, int ny);
        public String toString();
        public void translate(int dx, int dy);
}
```

Note

The drawing color can be set by two different methods:

setColor(Color c) in the class Graphics

setForeground(Color c) in the class Component

Doodle Example

Add a DrawPanel, an inner class, to a frame.

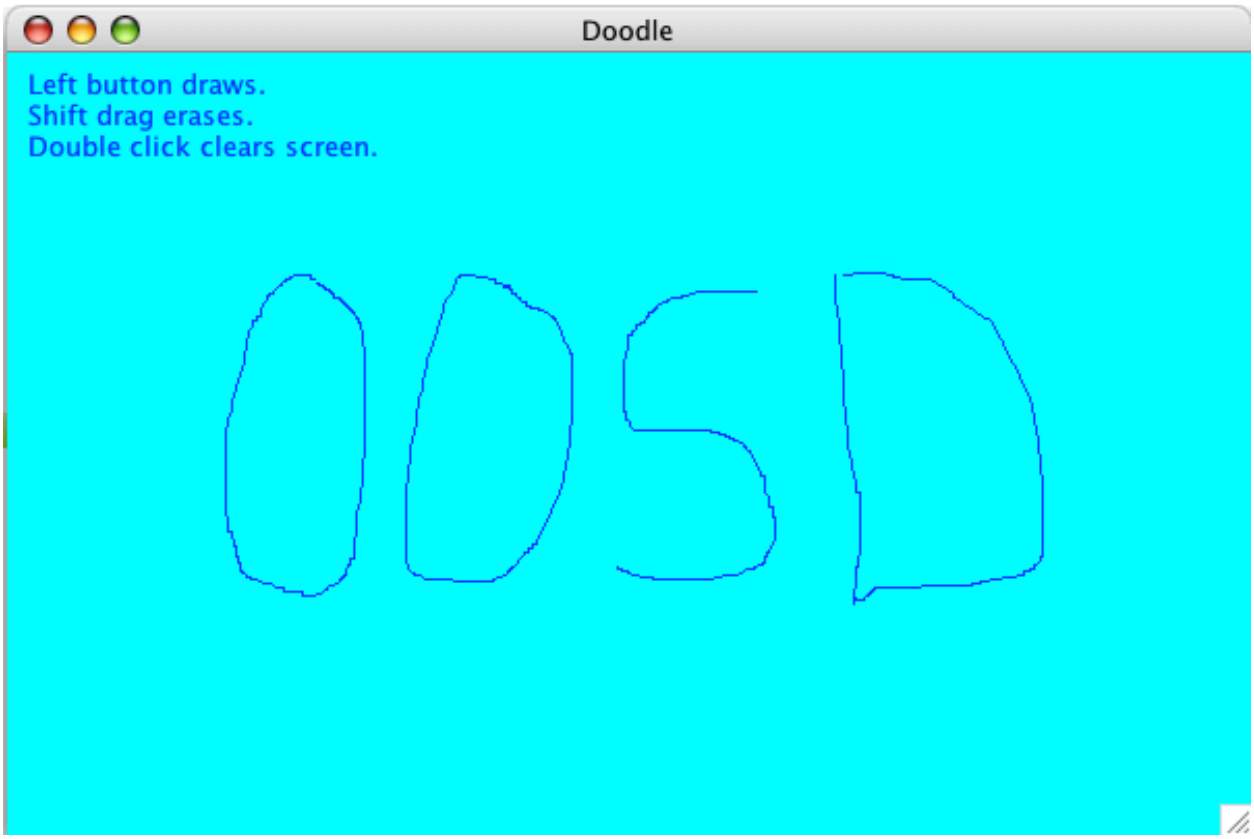
Capture mouse events to allow drawing on the panel.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Doodle extends JFrame
{
   private Point lineStart = new Point(0,0);
   private int size = 16;                  // erasing width
   private DrawPanel panel;

   Doodle()
   {
      setTitle("Doodle");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      addMouseListener(new Down());
      addMouseMotionListener(new Drag());
      panel = new DrawPanel();
      panel.setBackground(Color.cyan);
      getContentPane().add(panel);
   }

   public static void main(String [] a)
   {
      JFrame jf = new Doodle();
      jf.setSize(600, 400);
      jf.setVisible(true);
   }
```

```
class DrawPanel extends JPanel
{
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);
      g.setColor(Color.blue);
      g.drawString("Left button draws", 10, 20);
      g.drawString("Shift drag erases", 10, 35);
      g.drawString("Double click clears", 10, 50);
   }
}

class Down extends MouseAdapter
{
   public void mousePressed(MouseEvent e)
   {
      int x = e.getX(), y = e.getY();
      if (e.isShiftDown())
         setForeground(panel.getBackground());
      else
         setForeground(Color.blue);
      if (e.getClickCount() == 2)         // double click
         panel.repaint();
      else
         lineStart.move(x,y);
   }
}

class Drag extends MouseMotionAdapter
{
   public void mouseDragged(MouseEvent e)
   {
      int x = e.getX(), y = e.getY();
      Graphics g = panel.getGraphics();
      if (e.isShiftDown())
```



```
            g.fillOval(x-size/2, y-size/2, size, size);
        else
            g.drawLine(lineStart.x, lineStart.y, x, y);
        lineStart.move(x,y);
    }
  }
}
```



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)

Usage

```
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");
```

Note: The text area may need to be placed on a JScrollPane so that all of the lines can be seen.

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)
```


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.

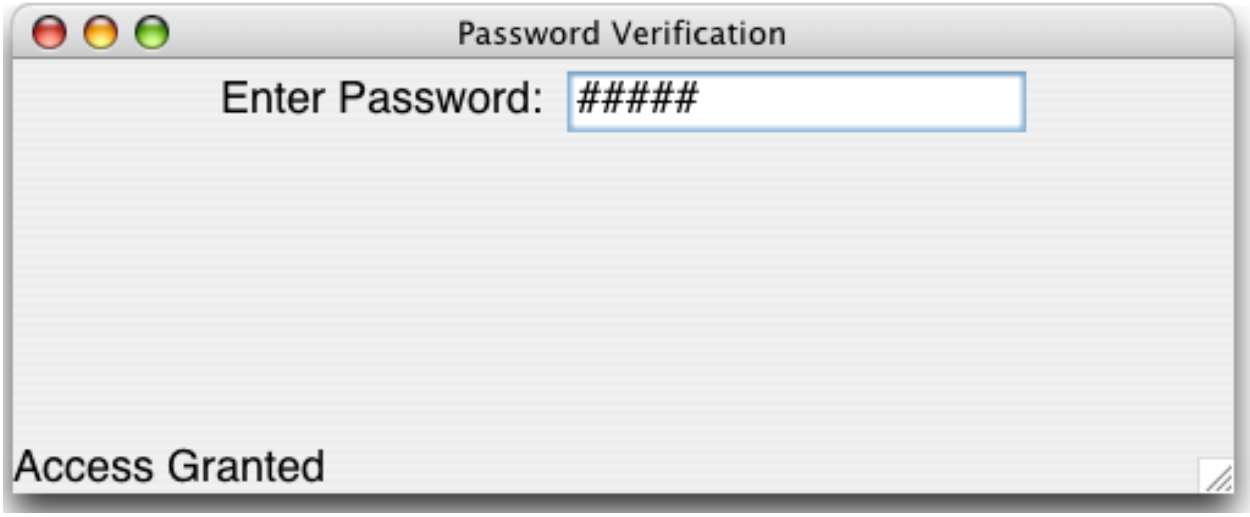
When return is typed in the field, it compares the string typed with a predefined password, “herky”.

```
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
    }
}
```



Component Hierarchy