Simplest Graphical Container

- An "option pane" is a simple graphical box that appears on screen to present a message or to request input from the user. The JOptionPane class has methods:

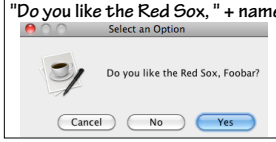
`showMessageDialog (parent, message)`	*displays message string in a box on the screen*
`showInputDialog (parent, message)`	*displays input box with message & returns user input as a String*
`showConfirmDialog (parent, message)`	*displays Yes/No/Cancel box with given message and returns one of 3 possible constants*

JOptionPane Examples


```
String name =
JOptionPane.showInputDialog
(null, "What's your name?");
```

see file DemoOptionPanes.java

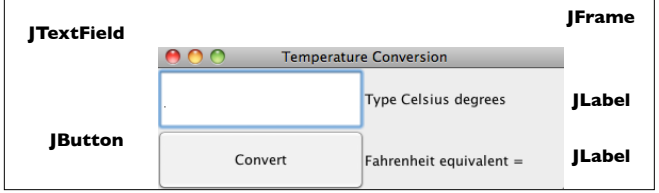
```
JOptionPane.showConfirmDialog(null,
"Do you like the Red Sox, " + name + "?");
```




```
JOptionPane.showMessageDialog
(null, "Of course! Who doesn't?");
```

Onscreen GUI Elements

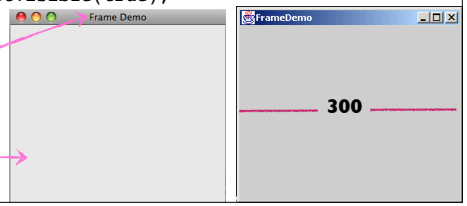
- Most GUIs are <u>not</u> composed of *option panes*; they are too limited. Instead, complex GUIs contain:
  - **frame**: A graphical window on the screen.
  - **components**: GUI widgets such as buttons or text fields.
  - **containers**: Logical groups of components.



Opening a "Window"

```
import javax.swing.JFrame;    // a subclass of java.awt.Frame

public class FrameTest
{
   public static void main(String[] args)
   {  JFrame frame = new JFrame ("Frame Demo");
      frame.setSize(300, 320);
      frame.setVisible(true);
   }
}
```



JFrames, continued

- An ins uch as a
  borde
  Applic
- Some J
  - Con
  - set ot)
  - set
  - set
  - set
  - set
    s

More on JFrames

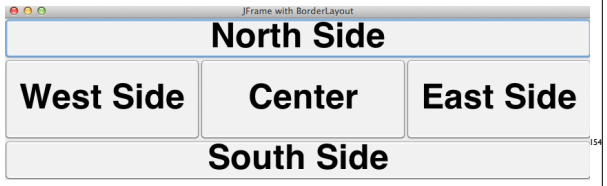
- In Java, a **Container** is a ***Component*** that can contain other ***Component***s. JFrames inherit methods from several superclasses:

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--java.awt.Window
                    |
                    +--java.awt.Frame
                          |
                          +--javax.swing.JFrame
```

- The hard work of organizing elements within a container is the task of the *layout manager*. It determines: the overall size of the container, the size of each element in the container, and the spacing and positioning of the elements.

java.awt.BorderLayout

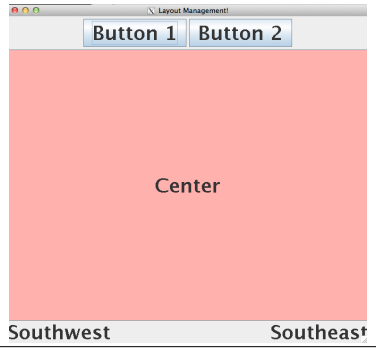
- Every container, by default, has a layout manager — an object that implements the LayoutManager interface — but you can easily replace it with another one.
  - **java.awt.BorderLayout**: arranges elements along the north, south, east, west, and center of the container. All extra space is placed in the center area. See program BorderLayoutDemo.java



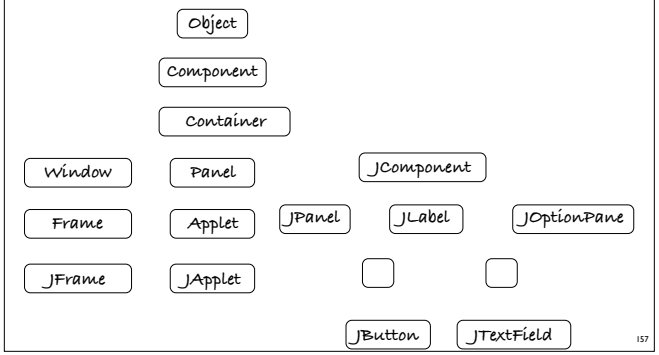
Other Layout Managers

- **javax.swing.BoxLayout**: arranges elements in 1 row or 1 column.
- **java.awt.CardLayout**: arranges elements like a stack of cards, with one visible at a time
- **java.awt.FlowLayout**: arranges left to right, top-bottom
- **java.awt.GridLayout**: arranges elements in a 2-dimensional grid of equally sized cells
- **java.awt.GridBagLayout**: arranges elements in a grid of variable-sized cells (complicated)
- Note: The Swing **`Border`** and **`BorderFactory`** classes can be used to put borders around almost any GUI element — this can make groupings of components more apparent and help guide (and inform) the user.
- See programs LayoutManagerDemo.java and GridBagLayoutDemo.java

Layout Mystery Problem



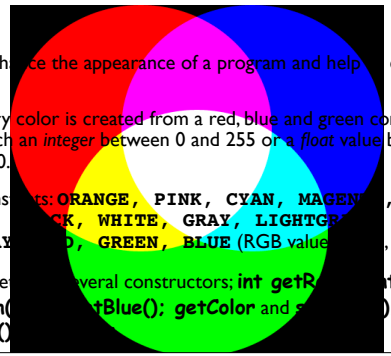
Partial Class Hierarchy



Methods of a Component

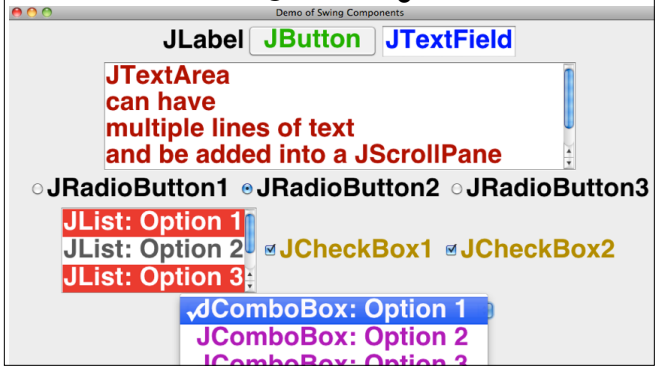
- A *component* (e.g., JButton) is an object having a graphical representation that can be displayed on the screen and can interact with the user.
  - `Color getForeground ()`
  - `Color getBackground ()`
  - `Font getFont()`
  - `void setBackground (Color c)`
  - `void setForeground (Color c)`
  - `void setFont (Font f)`
  - `boolean isVisible()`
  - `void setVisible (boolean b)`
  - `void setSize (int width, int height)`
  - `void setName (String name)`

Color Class: *See program* ColorDemo.java



- Colors enh ce the appearance of a program and help convey meaning.
- Note every color is created from a red, blue and green component (RGB), each an *integer* between 0 and 255 or a *float* value between 0.0 and 1.0.
- Color cons ts: **ORANGE, PINK, CYAN, MAGEN , YELLOW, CK, WHITE, GRAY, LIGHTGR DARKGRAY D, GREEN, BLUE** (RGB value , 255)
- **Color** Me veral constructors; **int getR t getGreen(tBlue(); getColor** and); **brighter()**

SwingDemo.java



Event-Driven Programming

- An "event" is an object representing a user's interaction with a GUI component (e.g., clicking on a component or moving the mouse over it).
- A "listener" is an object that is notified when an event occurs, and executes code to respond to that event. To organize event "listening," you use different event listener classes. For example,
  - An ActionEvent is generated from clicking a JButton, selecting a menu item in a JMenu, hitting the Enter key (or return) in a JTextField. To such components use addActionListener (instance of class that implements the ActionListener interface), which requires defining *void actionPerformed (ActionEvent e)*

Events and Listeners

- Mouse movement, mouse clicks & keystrokes cause different kinds of "events" to be generated. A Java program must install *event listener* objects in order to be notified about only certain of these events.
  - To listen to window events, add a "window listener" object to the frame. The **WindowListener** interface has 7 methods, or you can just implement certain of these methods via the **WindowAdapter** "convenience class."
  - A "mouse listener" must implement all of the **MouseListener** interface (5 methods). Or it can implement just certain methods via the **MouseAdapter** class.
  - When the user clicks a button, presses Return while typing in a text field, or chooses a menu item, an **ActionEvent** is generated. A listener must implement the **ActionListener** interface (which contains one method).

Event - Listener Examples

Event Type	Listener Type	Some Methods
Window Events (e.g., JFrame jf)	jf.addWindowListener (*object of class that implements Windowlistener or extends WindowAdapter*);	windowClosing windowIconified windowOpened *4 others*
Action Events (e.g., JButton jb)	jb.addActionListener (*object of class that implements ActionListener*);	actionPerformed
Mouse Events (e.g., Component c)	c.addMouseListener (*object of class that implements MouseListener or extends MouseAdapter*);	mouseEntered mouseClicked mouseExited mousePressed mouseReleased

Window Events

- A WindowEvent is generated when you
  - open a window for the first time,
  - close it,
  - iconify it,
  - activate it, etc.

- To such components, use addWindowListener(instance of class that implements the WindowListener interface or extends the WindowAdapter class) .
- To listen to window events, add a "window listener" object to the frame. The **WindowListener** interface has 7 methods; or one may implement certain of these methods via the **WindowAdapter** "convenience class."
  - See ActionEventDemo.java for an example

Implementing WindowListener Interface

- The first step in preparing an application to respond to Window events is to define a "listener" class that implements the WindowListener interface:

```
import java.awt.event.*;
public class WindowChatterBox  implements WindowListener
{
   public void windowClosing ( WindowEvent e )
    {  System.out.println ("Window got CLOSED!");  System.exit(0);  }
   public void windowActivated ( WindowEvent e )        {  }
   public void windowClosed ( WindowEvent e )           {  }
   public void windowDeactivated ( WindowEvent e )      {  }
   public void windowDeiconified ( WindowEvent e )      {  }
   public void windowIconified ( WindowEvent e )
      {  System.out.println ("Window got iconified!");  }
   public void windowOpened (WindowEvent e)             {  }
}
```

Some java.awt.event Classes

Components	Events	Description
Button, JButton	ActionEvent	User clicked button
CheckBox, JCheckBox	ItemEvent	User toggled a checkbox
ScrollBar, JScrollBar	AdjustmentEvent	User moved the scrollbar
Component, JComponent	ComponentEvent	Component was moved or resized
	FocusEvent	Component acquired or lost focus
	KeyEvent	User typed a key
TextField, JTextField	ActionEvent	User typed Enter key
Window, JWindow	WindowEvent	User manipulated window

Some javax.swing.event Classes

Components	Events	Description
JPopupMenu	PopupMenuEvent	User selected a choice
JComponent	AncestorEvent	An event occurred in an ancestor
JList	ListSelectionEvent	User double-clicked a list item
	ListDataEvent	List's contents were changed
JMenu	MenuEvent	User selected menu item
JTextComponent	CaretEvent	User clicked in text
JTable	TableModelEvent	Items added/removed from table
	TableColumnModelEvent	A table column was moved

[Anonymous] Inner Classes

- An inner class is one that's defined inside of another class. The syntax is ugly, but it provides a useful way of creating classes and objects "on the fly." The body of the class definition is put right after the **new** operator, as illustrated:

```
  …
JFrame jf = new JFrame ("test frame");
jf.setSize(400, 300);
jf.setVisible(true);
jf.addWindowListener   (
   new WindowAdapter()
    {
       public void windowClosing (WindowEvent e)
       {  System.exit(0);   }
    }                  );
  …
```

Listening to Mouse Events

- An event is a change in status that can initiate a responsive action. For example, when you click a mouse button, you change the status of the mouse, thus launching a mouse event.
- Mouse events can be trapped for any GUI widget that derives from Component. Each of the following takes a MouseEvent object as its argument (which has the x,y coordinates of where the event occurred).
  - The MouseListener methods are mousePressed, mouseClicked, mouseReleased, mouseEntered and mouseExited.
  - The MouseMotionListener methods are mouseDragged and mouseMoved.

The Graphics Object

- To draw shapes and lines, you need access to a Graphics object.
- This object is like a paintbrush; it can be dipped into any color, e.g., if g is the Graphics object:
  g.setColor (Color.RED);
- The Graphics object is usually obtained inside the paintComponent method of a JComponent or JPanel

g.fillRect (10, 30, 60, 35);
g.fillOval (80, 40, 50, 70);

How to Draw in a Window

- With Swing, the preferred method is to create a dedicated drawing area as a subclass of **`JPanel`**:

```
public class subclassName extends JPanel
{   …
   public void paintComponent (Graphics g)
    { // NEVER invoke paintComponent directly
       super.paintComponent( g );   // SOMETIMES needed
      // your drawing code goes here
         …
    }
}
```

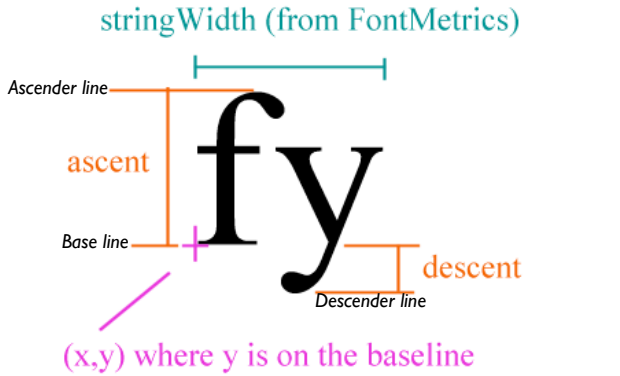
- If you draw on a Canvas, override the **`paint()`** method, not **`paintComponent`**.

Drawing Methods of a Graphics Object

- void **drawLine** (int x1, int y1, int x2, int y2)
- void **drawRect** (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 width, int height)
- void **clearRect** (int x, int y, int width, int height)
- void **drawOval** (int x, int y, int width, int height)
- void **fillOval** (int x, int y, int width, int height)
- void **drawArc** (int x, int y, int width, int height, int startA, int arcA)
- void **drawString** (String s, int x, int y)
- void **setFont** (Font f)
- void **setColor** (Color c)

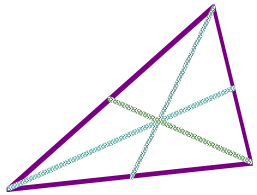
Fonts (see Surprise.java)

- To write a string, add a **drawString** statement to a paint or paintComponent method (defined in a subclass of **JPanel** or **Canvas**):
  - *graphicsContext*.**drawString (aString, x, y);**
- To determine the height of a font or width of a string, create an instance of the **FontMetrics** class and examine it's instance variables:
  - **FontMetrics** *var* = *graphicsContext*.**getFontMetrics();**
  - *var*.**getHeight ()**
  - *var*.**stringWidth (***aString***)**
- *graphicsContext*.setFont (
  new Font ("*fontName*", Font.*style*, *size*));



Testing a Theorem with Graphics

- We can write methods that operate on points and lines in such a way that the underlying algebra remains hidden. E.g., to test the theorem: "The medians of a triangle meet at a point"
- Suppose we have 4 arrays:
  - `Point [] vertices = new Point [3];`
  - `Point [] midpoints = new Point [3];`
  - `Line [] medians = new Line [3];`
  - `Point [] intersections = new Point [3];`



JSlider Controls

- JSliders enable the user to select from a range of integer values. When oriented horizontally, the minimum value is at the extreme left; vertical ones have the minimum value at the bottom.
  - The slider can show both major and minor tick marks between them. The # of values between the tick marks is controlled with setMajorTickSpacing(int n) & setMinorTickSpacing(int n) . Use setPaintTicks(true) .
- `JSlider (int orientation, int min, int max, int value)`
- `int getValue ()` *// Returns slider's current value.*
- `void setValue (int n)` *// Sets slider's current value.*

JSlider Controls, continued

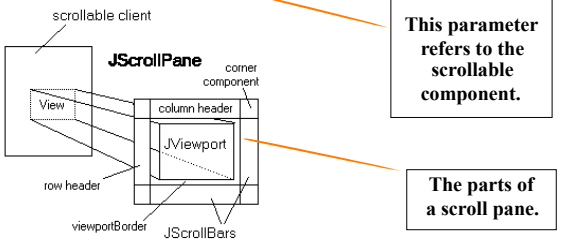
- JSliders generate ChangeEvents when the user interacts with the control. An object of a class that implements interface ChangeListener and defines method public void stateChanged (ChangeEvent e) can respond to ChangeEvents.

```
final JSlider s = new JSlider (
  SwingConstants.HORIZONTAL, 5, 200, 50);
s.addChangeListener ( new ChangeListener()
    { public void stateChanged(ChangeEvent e)
        {   System.out.println(s.getValue() ); }
    }
                        );        //see SliderTest.java
```

Scroll Panes

- A JScrollPane is an object that manages scrolling within a window or JTextArea.

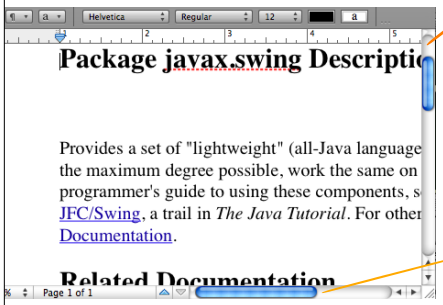
```
this.getContentPane().add(new JScrollPane(display));
```



This parameter refers to the scrollable component.

The parts of a scroll pane.

Scroll Pane Example



Scroll bars appear on the text area only when they are needed.

Horizontal scroll bar may or may not be necessary.

JTextAreas (see TextIO.java)

- A **JTextArea** is a multiline text area that can be used for either input or output. It is almost identical to the AWT component, TextArea, except it does not contain scrollbars by default.
  - `public JTextArea (String text, int rows, int columns)`
  - If jf is a defined JFrame, and jta is a defined JTextArea, then `jf.add (new JScrollPane(jta));` will add scrollbars to the text area.
- Additional useful methods:
  - `void append (String str)` `int getRows()`
  - `int getColumns()` `int getColumnWidth()`