An Introduction To Graphical User Interfaces

You will learn about the event-driven model and how to create simple graphical user interfaces (GUI's) in Java

Note: GUI Code Cannot Be Run Through A Text-Only Connection: SSH

```
[csb exampleTwo 45 ]> ls
Driver.class*      Driver.java       MyListener.class*  MyListener.java

[csb exampleTwo 46 ]> java Driver
Exception in thread "main" java.lang.InternalError: Can't connect to X11 window server
  using ':0.0' as the value of the DISPLAY variable.
        at sun.awt.X11GraphicsEnvironment.initDisplay(Native Method)
        at sun.awt.X11GraphicsEnvironment.<clinit>(X11GraphicsEnvironment.java:125)
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Class.java:140)
        at
  java.awt.GraphicsEnvironment.getLocalGraphicsEnvironment(GraphicsEnvironment.jav
  a:62)
        at java.awt.Window.init(Window.java:223)
        at java.awt.Window.<init>(Window.java:267)
        at java.awt.Frame.<init>(Frame.java:398)
        at java.awt.Frame.<init>(Frame.java:363)
        at Driver.main(Driver.java:7)
```

Components

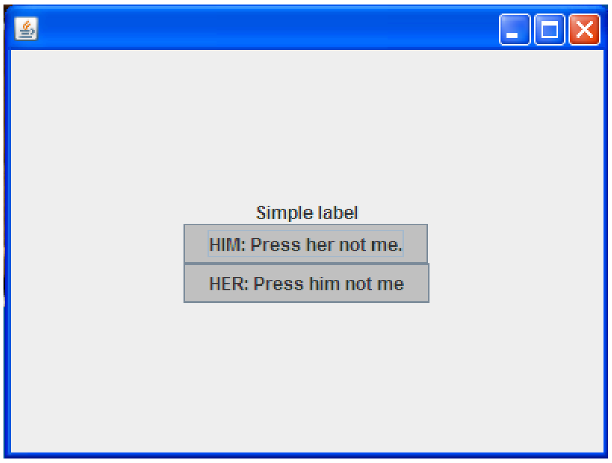
- They are many types of graphical controls and displays available:
  - JButton, JFrame, JLabel, JList, JTextArea, Window
- Also known as "widgets"
- For Sun's online documentation refer to the url:
  - *http://download.oracle.com/javase/7/docs/api/* (especially java.awt.event, javax.swing.event, and javax.swing).

Containers

- A special type of component that is used to hold/contain the components (subclass of the basic component class)
- Can be used to group components on the screen (i.e., one container holds another container which in turn groups a number of controls).
- You must have at least one container object for your GUI:
  - JPanel, JWindow, JDialog, JFrame
- Components which have been added to a container will appear/disappear and be garbage collected along with the container.

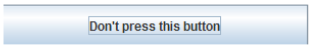
Containers

•You must have at least one container object for your GUI:
  - JPanel, JWindow, JDialog, **JFrame**

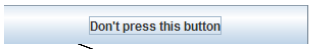


Some Relevant Java GUI libraries

1. Java classes for the components and containers
   - e.g., JButton class
   - javax.swing (import javax.swing.* or import javax.swing.<*class name*>)

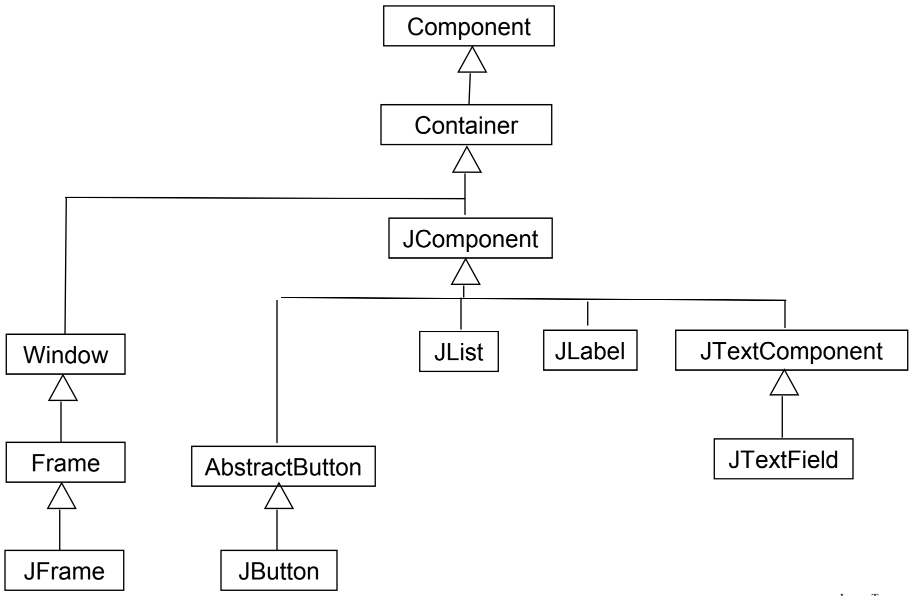


2. Java classes with the code to react to user-initiated events
   - e.g., code to react when a button is pressed
   - java.awt.event (import java.awt.event.*, import javax.swing.event.*)

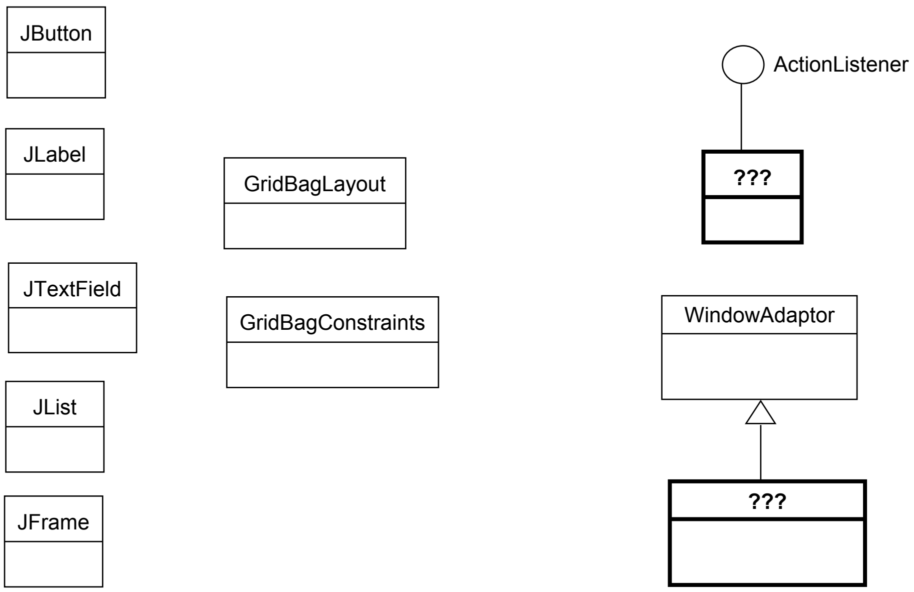


```
class ButtonListener implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
            :           :           :
    }
}
```

Hierarchy: Important Widget Classes



Some Relevant Java GUI Classes For This Section



Traditional Software

Program control is largely determined by the program through a series of sequential statements.

Example

```
        :
        if (num >= 0)
        {
            // Statements for the body of the if
        }
        else
        {
            // Statements for the body of the else
        }
```

When num is non-negative

Num is negative

Traditional Software

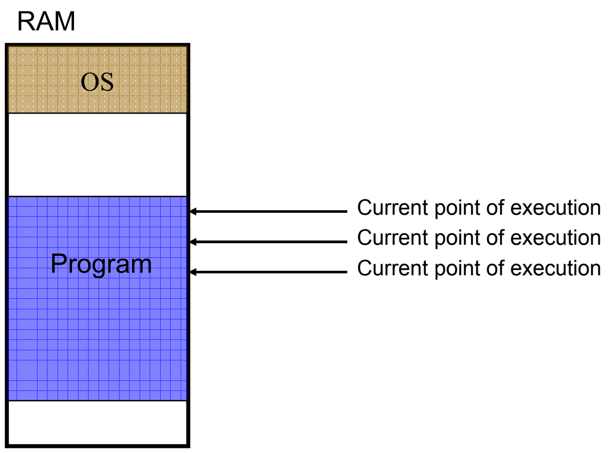
The user can only interact with the program at places that are specified by the program (e.g., when an input statement is encountered).

Example

```
    Scanner aScanner = new Scanner (System.in);
    System.out.print("Enter student ID number: ");
    id = aScanner.nextInt ();
```

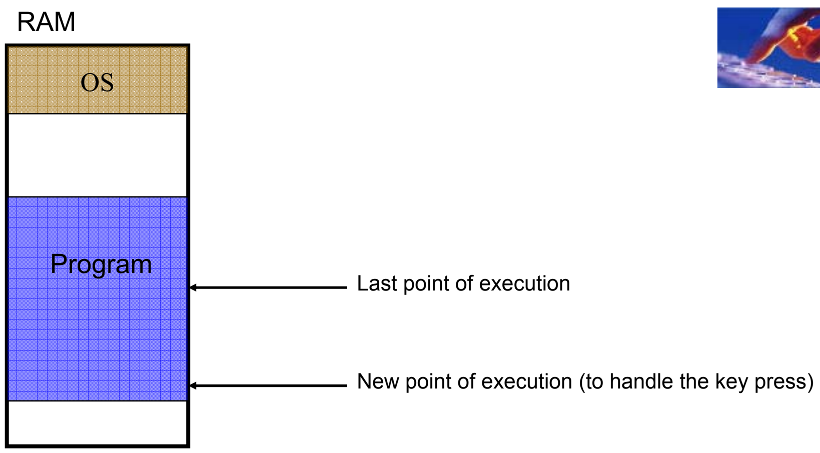
Event-Driven Software

Program control can also be sequential


RAM
OS
Program
Current point of execution
Current point of execution
Current point of execution

Event-Driven Software

In addition program control *can also* be determined by events

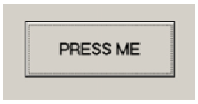


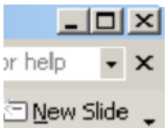
Characteristics Of Event Driven Software

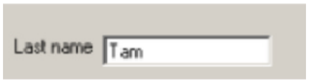
- Program control can be determined by events as well as standard program control statements.
- A typical source of these events is the user.
- These events can occur at any time.

Most Components Can Trigger Events

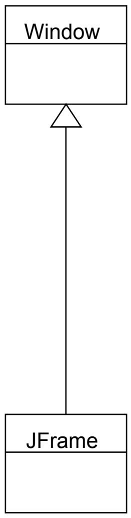
- Graphical objects can be manipulated by the user to trigger events.
- Each graphical object can have 0, 1 or many events that can be triggered.

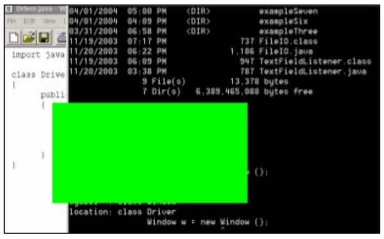






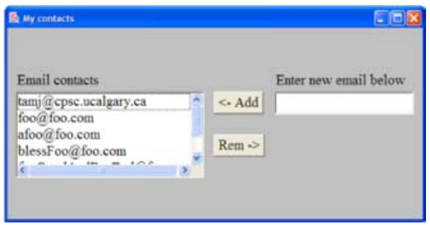
Window Classes





```
import java
class Drive
{
    publi
    {

    }
}
```

```
04/01/2004  05:00 PM    <DIR>          exampleSeven
04/01/2004  04:09 PM    <DIR>          exampleSix
03/31/2004  06:58 PM    <DIR>          exampleThree
11/19/2003  07:17 PM               737 FileIO.class
11/20/2003  06:22 PM             1,186 FileIO.java
11/19/2003  06:09 PM               947 TextFieldListener.class
11/20/2003  03:38 PM               787 TextFieldListener.java
               9 File(s)         13,378 bytes
               7 Dir(s)   6,389,465,088 bytes free
                                        ();
location: class Driver
                Window w = new Window ();
```



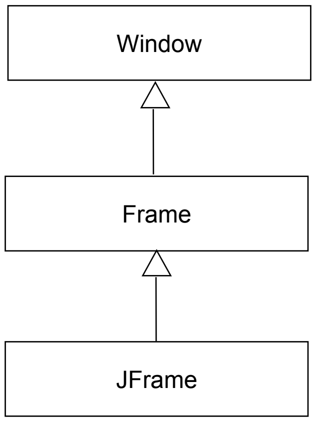
My contacts

Email contacts

tamj@cpsc.ucalgary.ca
foo@foo.com
afoo@foo.com
blessFoo@foo.com

Enter new email below

The Window Class Hierarchy



Class JFrame

For full details look at the online API:

- http://download.oracle.com/javase/6/docs/api/javax/swing/JFrame.html

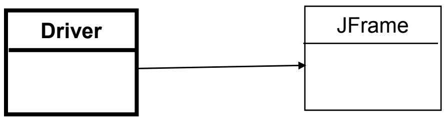
Some of the more pertinent methods:

- JFrame ("<*Text on the title bar*>")
- setSize (<*pixel width*>, <*pixel height*>)
- setVisible (<*true/false*>)
- setDefaultCloseOperation (<*class constants*>[1])

1 DISPOSE_ON_CLOSE, HIDE_ON_CLOSE, DO_NOTHING_ON_CLOSE

Example: Creating A Frame That Can Close (And Cleanup Memory After Itself)

The complete code for this example can be found in UNIX under the path:
/home/233/examples/gui/first_frame



```
import javax.swing.*;
public class Driver
{
    public static void main (String [] args)
    {
        JFrame mf = new JFrame ("Insert title here");
        mf.setSize (300,200);
        mf.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        mf.setVisible(true);
    }
}
```

Pitfall 1: Showing Too Early

- When a container holds a number of components the components must be added to the container (later examples).
- To be on the safe side the call to the "setVisible()" method should be done after the contents of the container have already been created and added.

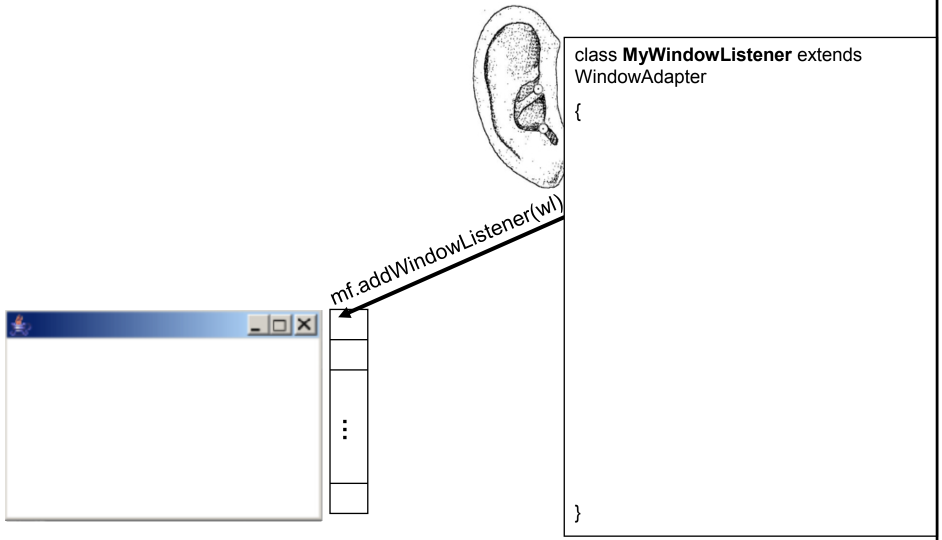
Window Events

- The basic JFrame class provides basic capabilities for common windowing operations: minimize, maximize, resize, close.
- However if a program needs to perform other actions (i.e., your own custom code) when these events occur the built in approach won't be sufficient.
  - E.g., the program is to automatically save your work to a file when you close the window.

Steps In The Event Model For Handling A Frame Event: Window Closing

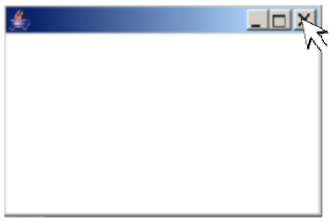
1) The frame must register all interested event listeners.
2) The user triggers the event by closing the window
3) The window sends a message to all listeners of that event.
4) The window event listener runs the code to handle the event (e.g., save information to a file).

1. The Frame Must Register All Interested Event Listeners.



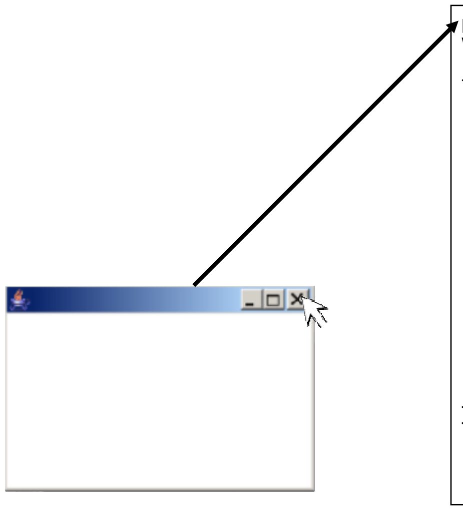
```
class MyWindowListener extends
WindowAdapter
{

}
```

2. The User Triggers The Event By Closing The Window



James Tam

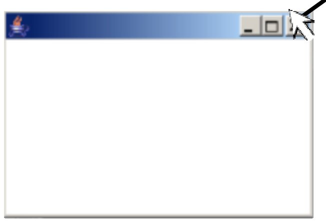
3. The Window Sends A Message To All Listeners Of That Event.



```
public class MyWindowListener extends
WindowAdapter
{
        public void windowClosing  (WindowEvent e)
        {



        }
}
```

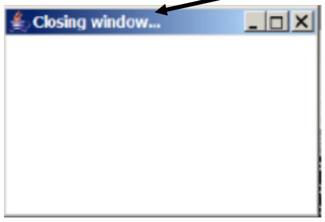
James Tam

4. The Event Listener Runs The Code To Handle The Event.



```
public class MyWindowListener extends
WindowAdapter
{
      public static final int DELAY_LENGTH =
                                  500000000;
        public void windowClosing (WindowEvent e)
        {
            JFrame aFrame = (JFrame) e.getWindow();
            aFrame.setTitle("Closing window...");
            for (int i = 0; i < DELAY_LENGTH; i++);
               aFrame.setVisible(false);
            aFrame.dispose();
        }
}
```

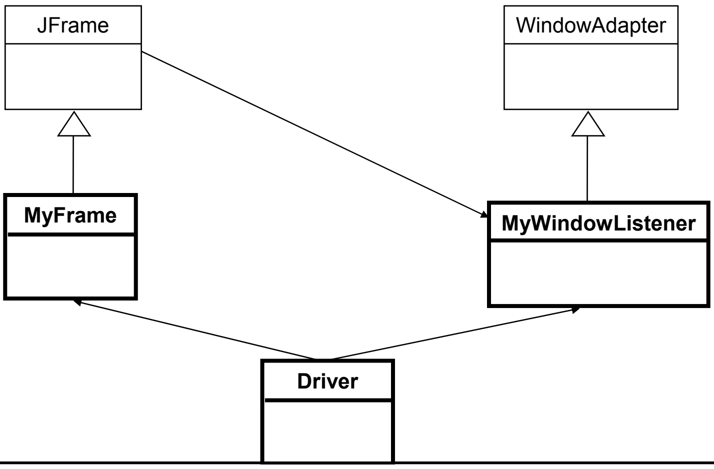
4. The Event Listener Runs The Code To Handle The Event.



```
public class MyWindowListener extends
WindowAdapter
{
 public static final int DELAY_LENGTH =
                                  500000000;
        public void windowClosing (WindowEvent e)
        {
            JFrame aFrame = (JFrame) e.getWindow();
            aFrame.setTitle("Closing window...");
            for (int i = 0; i < DELAY_LENGTH; i++);
                aFrame.setVisible(false);
            aFrame.dispose();
        }
}
```

An Example Of Handling A Frame Event

The complete code for this example can be found in UNIX under the path: /home/233/examples/gui/second_window_events



The Driver Class

```
import javax.swing.JFrame;

public class Driver
{
    public static final int WIDTH = 300;
    public static final int HEIGHT = 200;
    public static void main (String [] args)
    {
        MyFrame aFrame = new MyFrame ();
        MyWindowListener aListener = new MyWindowListener() ;
        aFrame.addWindowListener(aListener);
        aFrame.setSize (WIDTH,HEIGHT);
        aFrame.setVisible(true);
    }
}
```

Class MyFrame

```
import javax.swing.JFrame;

public class MyFrame extends JFrame
{
    // More code will be added in later examples.
}
```

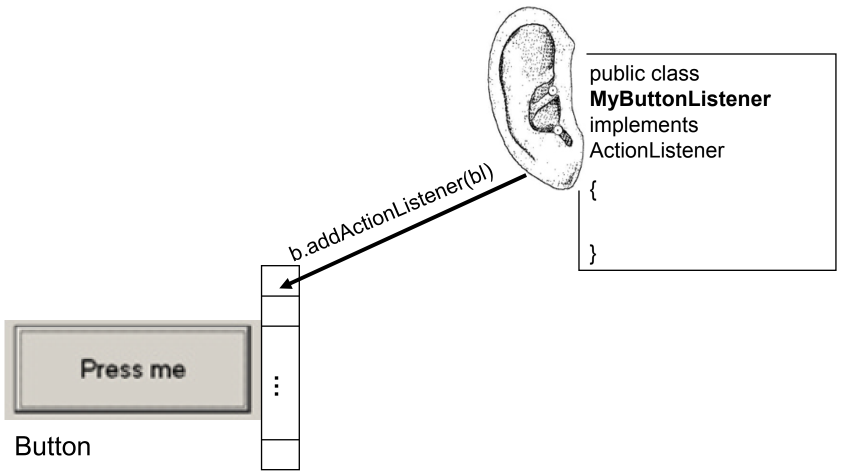
Class MyWindowListener

```
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFrame;

public class MyWindowListener extends WindowAdapter
{
        public static final int DELAY_LENGTH = 500000000;
        public void windowClosing (WindowEvent e)
        {
          JFrame aFrame = (JFrame) e.getWindow();
          aFrame.setTitle("Closing window...");
          for (int i = 0; i < DELAY_LENGTH; i++);
             aFrame.setVisible(false);
          aFrame.dispose();
        }
}
```

Steps In The Event Model For Handling A Button Event

1) The button must register all interested event listeners.
2) The user triggers an event by pressing a button.
3) The button sends a message to all listeners of the button press event.
4) The button listener runs the code to handle the button press event.

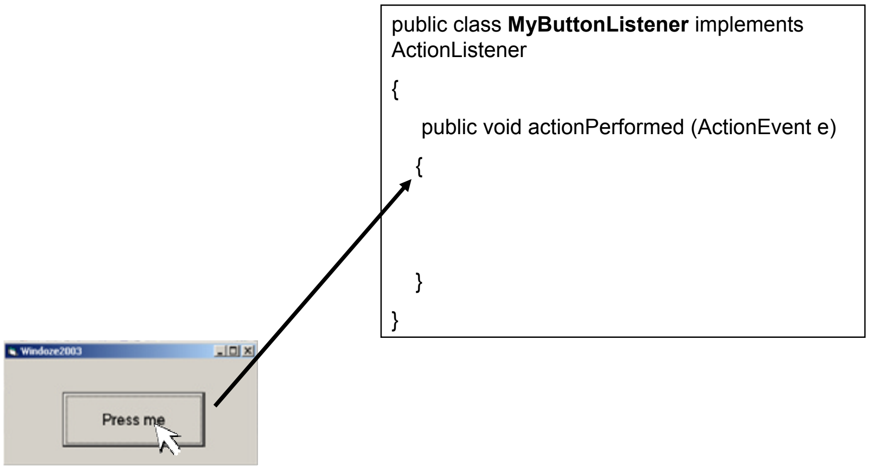
1. The Graphical Component Must Register All Interested Event Listeners.



```
public class
MyButtonListener
implements
ActionListener
{

}
```

2. The User Triggers An Event By Pressing The Button

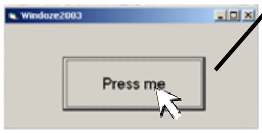
3. The Component Sends A Message To All Registered Listeners For That Event



```
public class MyButtonListener implements
ActionListener
{
    public void actionPerformed (ActionEvent e)
    {


    }
}
```

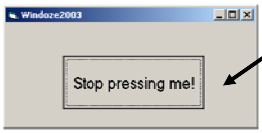
3. The Component Sends A Message To All Registered Listeners For That Event

```
public class MyButtonListener implements
ActionListener
{
    public void actionPerformed (ActionEvent e)
    {
        JButton b = (JButton) e.getSource();
        b.setLabel("Stop pressing me!");
    }
}
```



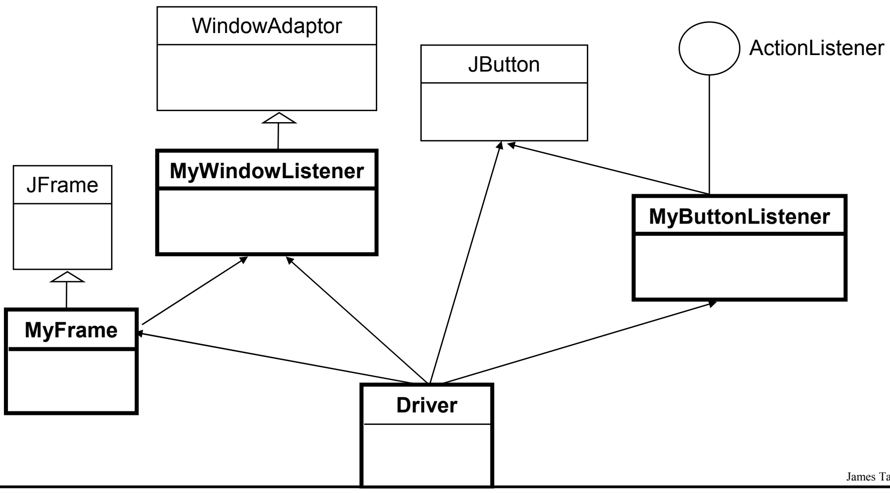
4. The Event Listener Runs The Code To Handle The Event

```
public class MyButtonListener implements
ActionListener
{
    public void actionPerformed (ActionEvent e)
    {
        JButton b = (JButton) e.getSource();
        b.setLabel("Stop pressing me!");
    }
}
```



An Example Of Handling A Button Event

The complete code for this example can be found in UNIX under the path: /home/233/examples/gui/three_button_events



An Example Of Handling A Button Event: The Driver Class

```
import javax.swing.JButton;

public class Driver
{
    public static final int WIDTH = 300;
    public static final int HEIGHT = 200;
    public static void main (String [] args)
    {
        MyFrame aFrame = new MyFrame ();
        MyWindowListener aWindowListener = new MyWindowListener();
        aFrame.addWindowListener(aWindowListener);
        aFrame.setSize (WIDTH,HEIGHT);
```

An Example Of Handling A Button Event: The Driver Class (2)

```
        JButton aButton = new JButton("Press me.");
        MyButtonListener aButtonListener = new
        MyButtonListener();
        aButton.addActionListener(aButtonListener);
        aFrame.add (aButton);
        aFrame.setVisible(true);
    }
}
```

An Example Of Handling A Button Event: The ButtonListener Class

```
import javax.swing.JButton;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class MyButtonListener implements ActionListener
{
   public void actionPerformed (ActionEvent e)
   {
        JButton aButton = (JButton) e.getSource();
        aButton.setText("Stop pressing me!");
   }
}
```

Anonymous Objects/Anonymous Inner Class

- If an object needs to be created but never directly referenced then it may be candidate for being created as an anonymous object.
- An example of where an anonymous object may be created is an event listener.

No reference name

```
JButton aButton = new JButton("Press me.");
aButton.addActionListener (new ActionListener() {
                public void actionPerformed(ActionEvent e)  {
    JButton aButton = (JButton) e.getSource();
    aButton.setText("Stop pressing me!");
  }
 });
```

Awkward if complex programming is required.

Nested/Inner Classes

- Occurs when one class is defined inside of another class:

```
public class X
{
    private class Y
    {

    }
}
```

Outer class

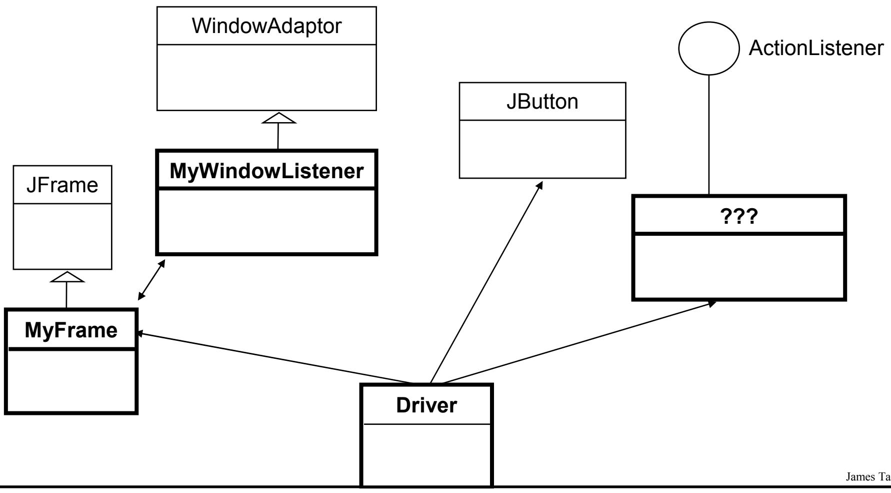
Inner class

- Why nest class definitions[1]:
  - It is a way of logically grouping classes that are only used in one place.
  - Nested classes can lead to more readable and maintainable code.
  - It increases encapsulation.
- Similar to declaring anonymous objects, nesting classes may be used when creating event listeners.

1 For more information: http://download.oracle.com/javase/tutorial/java/javaOO/nested.html

Alternate Example For Handling Events (Better)

The complete code for this example can be found in UNIX under the path: /home/233/examples/gui/four_button_alternate



The Driver Class

```
import javax.swing.JButton;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Driver
{
    public static final int WIDTH = 300;
    public static final int HEIGHT = 200;
    public static void main (String [] args)
    {
        MyFrame aFrame = new MyFrame ();
        aFrame.setSize (WIDTH,HEIGHT);
        JButton aButton = new JButton("Press me.");
```

The Driver Class (2)

```
        // Anonymous object/class
        aButton.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
          JButton aButton = (JButton) e.getSource();
          aButton.setText("Stop pressing me!");
         }
        });
        aFrame.add(aButton);
        aFrame.setVisible(true);
    }
}
```

Class MyFrame

```
import javax.swing.JFrame;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class MyFrame extends JFrame
{
    public MyFrame ()
    {
        MyWindowListener aWindowListener = new MyWindowListener();
        this.addWindowListener(aWindowListener);
    }
```

Class MyFrame (2)

```
    // Inner class defined within the MyFrame class.
    // Private because it's only used by the MyFrame class.
    private class MyWindowListener extends WindowAdapter
    {
      public static final int DELAY_LENGTH = 500000000;
      public void windowClosing (WindowEvent e)
      {
        JFrame aFrame = (JFrame) e.getWindow();
        aFrame.setTitle("Closing window...");
        for (int i = 0; i < DELAY_LENGTH; i++);
          aFrame.setVisible(false);
        aFrame.dispose();
      }
    }
}
```

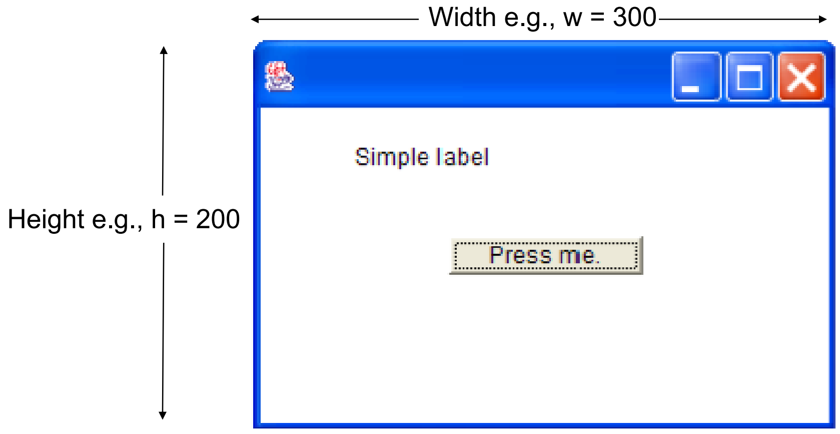
How To Handle The Layout Of Components

1. Manually set the coordinates yourself
2. Use one of Java's built-in layout manager classes

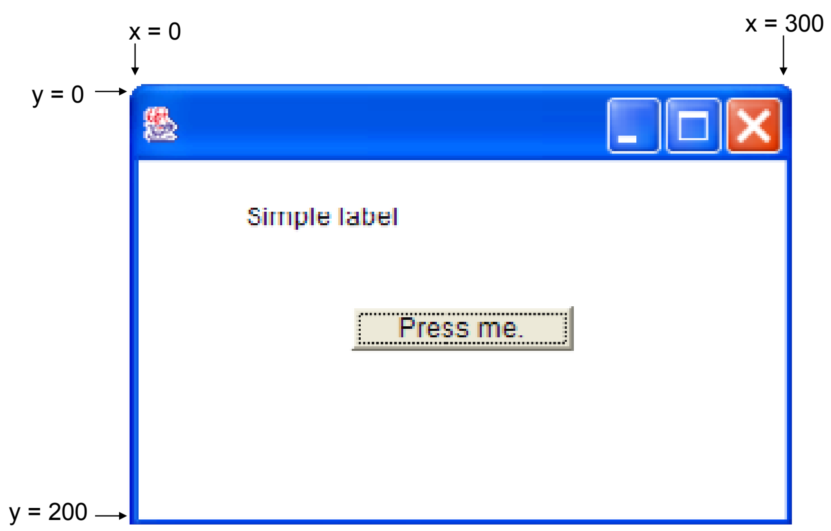
Layout Is Based On Spatial Coordinates

e.g. MyFrame my =new MyFrame ();

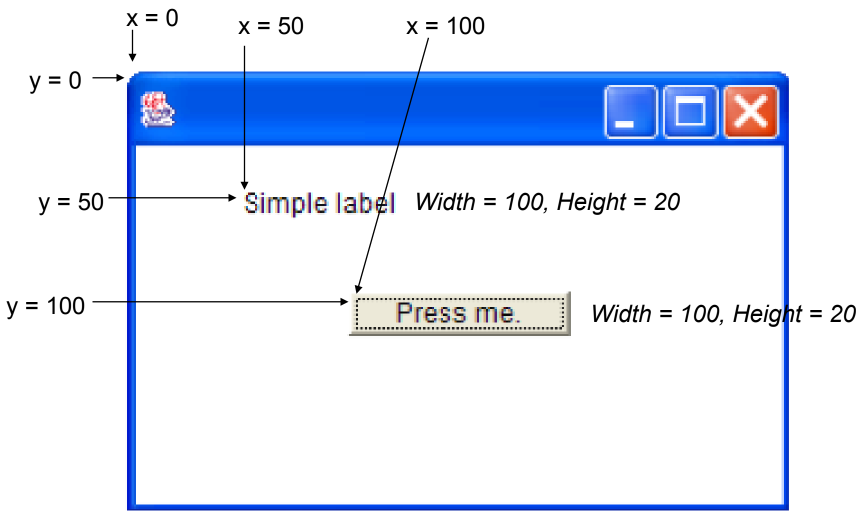
my.setSize(300,200);



Layout Is Based On Spatial Coordinates



Coordinates Of Components: Relative To The Container



Pitfall 2: Invisible Component

- Don’t forget that coordinates (0,0) are covered by the title bar of the frame.
- Components added at this location may be partially or totally hidden by the title bar.

A Example Showing Manual Layout

The complete code for this example can be found in UNIX under the path:

/home/233/examples/gui/fifth_manual_layout

An Example Showing Manual Layout: The Driver Class

```
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JFrame;

public class Driver
{
    public static final int WIDTH_FRAME = 300;
    public static final int HEIGHT_FRAME = 300;
    public static final int X_COORD_BUTTON = 100;
    public static final int Y_COORD_BUTTON = 100;
    public static final int WIDTH_BUTTON = 100;
    public static final int HEIGHT_BUTTON = 20;
    public static final int X_COORD_LABEL = 50;
    public static final int Y_COORD_LABEL = 50;
    public static final int WIDTH_LABEL = 100;
    public static final int HEIGHT_LABEL = 20;
```

An Example Showing Manual Layout: The Driver Class (2)

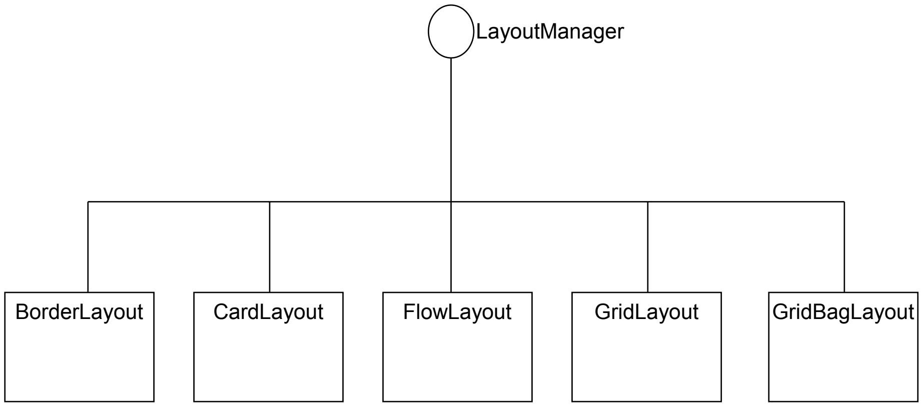
```
    public static void main (String [] args) {
        JFrame aFrame = new JFrame ();
        aFrame.setLayout(null);
        aFrame.setSize (WIDTH_FRAME,HEIGHT_FRAME);
        JButton aButton = new JButton("Press me.");
        aButton.setBounds(X_COORD_BUTTON,
                          Y_COORD_BUTTON,
                          WIDTH_BUTTON,
                          HEIGHT_BUTTON);
        JLabel aLabel = new JLabel ("Simple label");
        aLabel.setBounds(X_COORD_LABEL,
                         Y_COORD_LABEL,
                         WIDTH_LABEL,
                         HEIGHT_LABEL);
        aFrame.add(aButton);
        aFrame.add(aLabel);
        aFrame.setVisible(true);
    }
}
```

How To Handle The Layout Of Components

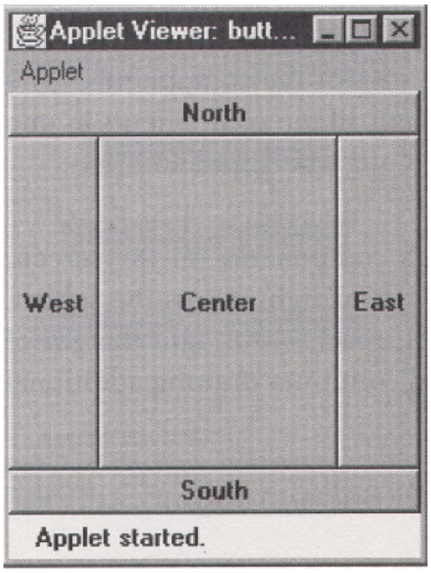
1. Manually set the coordinates yourself
2. **Use one of Java's built-in layout manager classes**

Java Layout Classes

There are many implementations (this diagram only includes the original classes that were implemented by Sun).

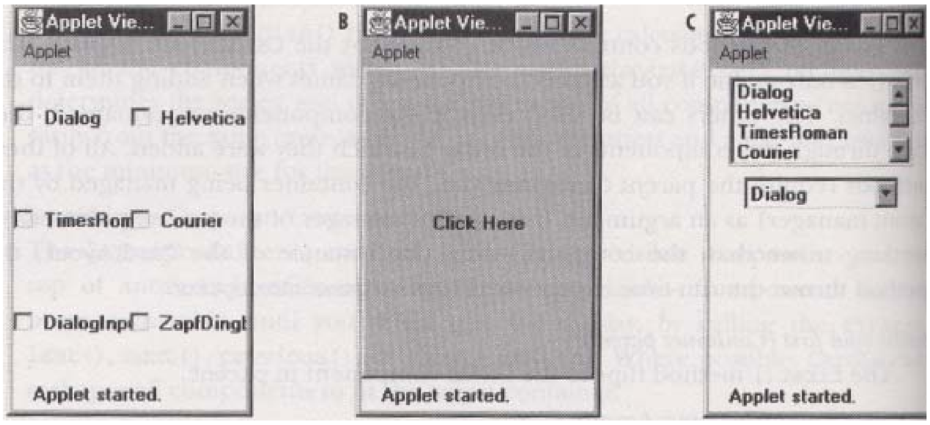


BorderLayout



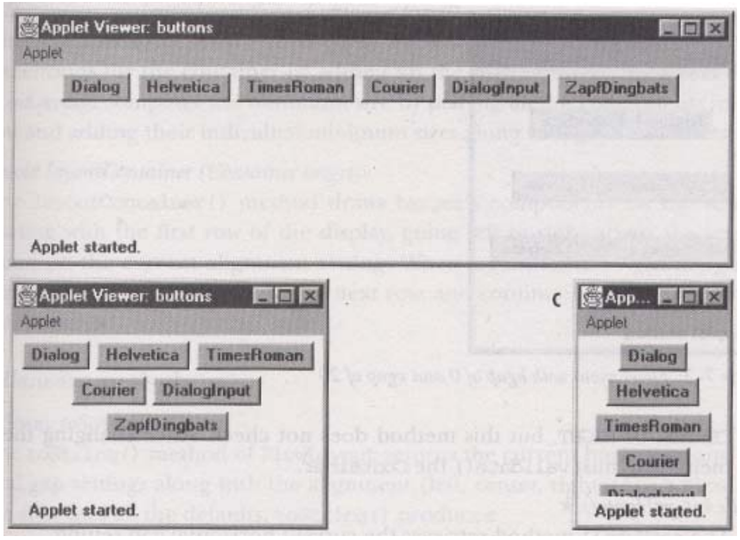
From Java: AWT Reference p. 256

CardLayout



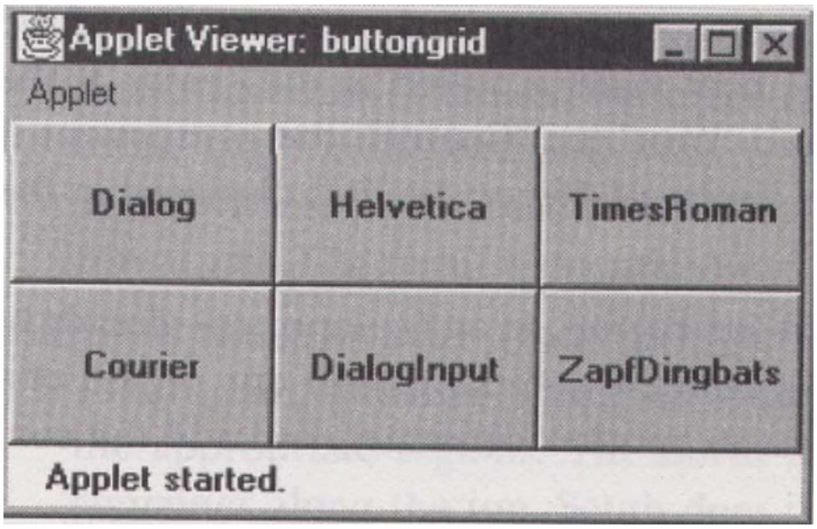
From Java: AWT Reference p. 264

FlowLayout



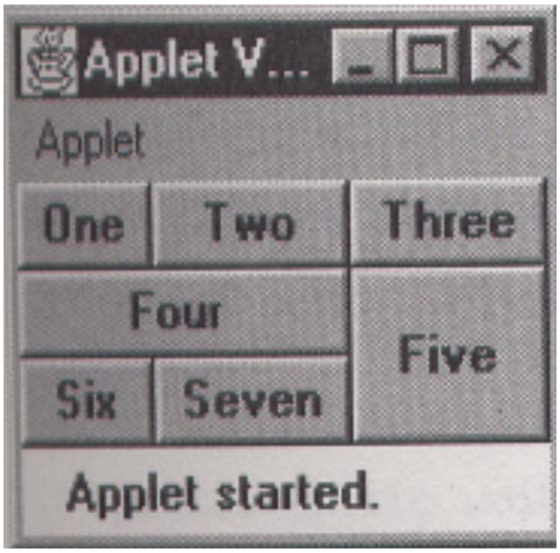
From Java: AWT Reference p. 253

GridLayout



From Java: AWT Reference p. 260

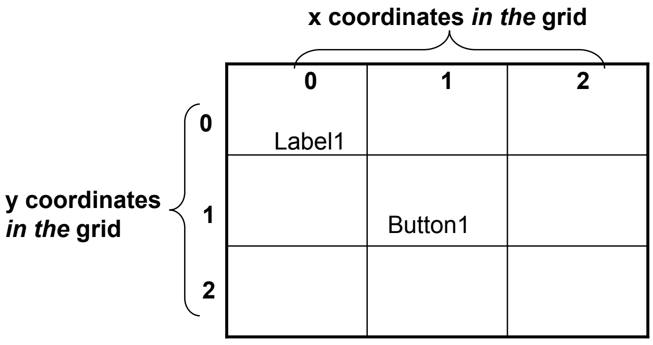
GridBagLayout



From Java: AWT Reference p. 269

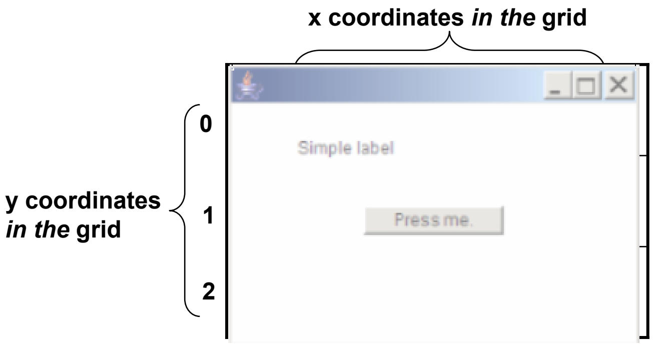
Designing A GUI When Using The GridBagLayout

Use graph paper or draw out a table.



Designing A GUI When Using The GridBagLayout

Use graph paper or draw out a table.



GridBagConstraints

•Goes with the GridBagLayout class.

•Because the GridBagLayout doesn't know how to display components you also need GridBagConstraints to constrain things (determine the layout).

•GridBagConstraints indicates how components should be displayed within the GridBag.

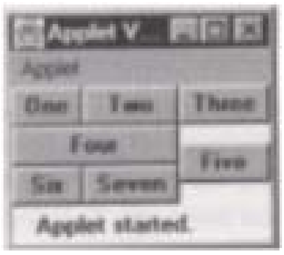
•For more complete information see:

  -*http://java.sun.com/javase/7/docs/api/java/awt/GridBagConstraints.html*

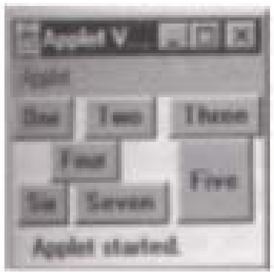
Some Important Parts Of The GridBagConstraints Class

```
public class GridBagConstraints
{
 // Used in conjunction with the constants below to determine the resize policy
 of the component
 public int fill;

 // Apply only if there is available space.
 // Determine in which direction (if any) that the component expands to fill the
 // space.
 public final static int NONE;
 public final static int BOTH;
 public final static int HORIZONTAL;
 public final static int VERTICAL;
```

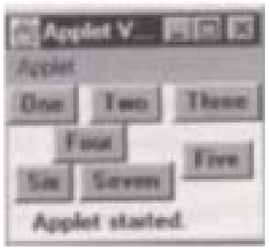
GridBagContraints: Fill Values



Horizontal



Vertical



None

Some Important Parts Of The GridBagConstraints Class (2)

```
// Position within the grid
public int gridx;
public int gridy;

// Number of grid squares occupied by a component
public int gridwidth;
public int gridheight;
```

Some Important Parts Of The GridBagConstraints Class (3)

```
// Used in conjunction with the constants below to determine that the
// component drift if the space available is larger than the component.
public int anchor;

// Apply only if the component is smaller than the available space.
// Determine in which direction that the component will be anchored there
public final static int CENTER;
public final static int EAST;
public final static int NORTH;
public final static int NORTHEAST;
public final static int NORTHWEST;
public final static int SOUTH;
public final static int SOUTHEAST;
public final static int SOUTHWEST;
public final static int WEST;
```

An Example Using The GridBagLayout

The complete code for this example can be found in UNIX under the path: /home/233/examples/gui/sixth_gridbaglayout

An Example Using The GridBagLayout: The Driver Class

```
public class Driver
{
    public static final int WIDTH = 400;
    public static final int HEIGHT = 300;
    public static void main (String [] args)
    {
        MyFrame aFrame = new MyFrame ();
        aFrame.setSize(WIDTH,HEIGHT);
        aFrame.setVisible(true);
    }
}
```

An Example Using The GridBagLayout: Class MyFrame

```
public class MyFrame extends JFrame
{
   private JButton aButton;
   private JLabel aLabel;
   private GridBagLayout aLayout;
   GridBagConstraints aConstraint;

   public MyFrame ()
   {
      MyWindowListener aWindowListener = new MyWindowListener ();
      addWindowListener(aWindowListener); // Calling method of super class.

      aConstraint = new GridBagConstraints();
      aButton = new JButton("Press me");
```

An Example Using The GridBagLayout: Class MyFrame (2)

```
    // Anonymous object.
    aButton.addActionListener (new ActionListener ()
    {
      public void actionPerformed (ActionEvent e)
      {
        JButton aButton = (JButton) e.getSource();
        aButton.setText("Stop pressing me!");
      }
    });
    aLabel = new JLabel("Simple label");
    aLayout = new GridBagLayout();
    setLayout(aLayout);     // Calling method of super class.
    addWidget(aLabel, 0, 0, 1, 1);
    addWidget(aButton, 2, 2, 1, 1);
} // End constructor
```

An Example Using The GridBagLayout: Class MyFrame (3)

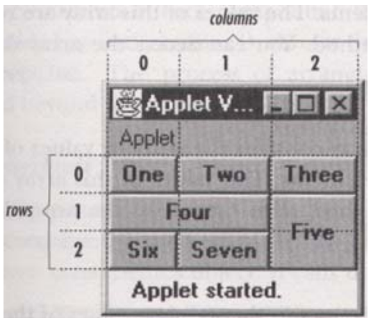
```
public void addWidget (Component widget, int x, int y, int w, int h)
{
    aConstraint.gridx = x;
    aConstraint.gridy = y;
    aConstraint.gridwidth = w;
    aConstraint.gridheight = h;
    aLayout.setConstraints (widget, aConstraint);
    add(widget);        // Calling method of super class.
}
```

An Example Using The GridBagLayout: Class MyFrame (3)

```
    // Innner class
    private class MyWindowListener extends WindowAdapter
    {
       public void windowClosing (WindowEvent e)
       {
         JFrame f = (JFrame) e.getWindow();
         f.setTitle("Closing window...");
         for (int i = 0; i < 500000000; i++);
             f.setVisible(false);
         f.dispose();
       }
    } // End of inner class definition
} // End of definition for class MyFrame
```

James Tam

Advanced Uses Of GridBagLayout



Button	gridx (col)	gridy (row)	grid-width	grid-height
One	0	0	1	1
Two	1	0	1	1
Three	2	0	1	1
Four	0	1	2	1
Five	2	1	1	2
Six	0	2	1	1
Seven	1	2	1	1

From Java: AWT Reference p. 269

James Tam

Components Effecting The State Of Other Components

The complete code for this example can be found in UNIX under the path:
/home/courses/219/examples/gui/seventh_controls_affect_controls

Components Effecting The State Of Other Components: The Driver Class

```
public class Driver
{
    public static final int WIDTH = 400;
    public static final int HEIGHT = 300;
    public static void main (String [] args)
    {
         MyFrame aFrame = new MyFrame ();
         aFrame.setSize(WIDTH,HEIGHT);
         aFrame.setVisible(true);
    }
}
```

Components Effecting The State Of Other Components: Class MyFrame

```
public class MyFrame extends JFrame
{
    private JLabel aLabel;
    private GridBagLayout aLayout;
    private GridBagConstraints aConstraint;
    private JButton himButton;
    private JButton herButton;
    private MyButtonListener aButtonListener;
```

Components Effecting The State Of Other Components: Class MyFrame (2)

```
public MyFrame ()
{
    MyWindowListener aWindowListener = new MyWindowListener ();
    addWindowListener(aWindowListener); // Calling method of super class.
    aConstraint = new GridBagConstraints();
    aButtonListener = new MyButtonListener();

    himButton = new JButton("HIM: Press her not me.");
    himButton.setActionCommand("him");
    himButton.setBackground(Color.lightGray);
    himButton.addActionListener(aButtonListener);

    herButton = new JButton("HER: Press him not me");
    herButton.setActionCommand("her");
    herButton.setBackground(Color.lightGray);
    herButton.addActionListener(aButtonListener);
```

Components Effecting The State Of Other Components: Class MyFrame (3)

```
    aLabel = new JLabel("Simple label");
    aLayout = new GridBagLayout();
    setLayout(aLayout);     // Calling method of super class.
    addWidget(aLabel, 0, 0, 1, 1);
    addWidget(himButton, 0, 1, 1, 1);
    addWidget(herButton, 0, 2, 1, 1);
}
```

Components Effecting The State Of Other Components: Class MyFrame (4)

```
public void addWidget (Component widget, int x, int y, int w, int h)
{
    aConstraint.gridx = x;
    aConstraint.gridy = y;
    aConstraint.gridwidth = w;
    aConstraint.gridheight = h;
    aLayout.setConstraints (widget, aConstraint);
    add(widget);        // Calling method of super class.
}

public JButton getHerButton () { return herButton; }
public JButton getHimButton () { return himButton; }
```

Components Effecting The State Of Other Components: Class MyFrame (5)

```
    private class MyWindowListener extends WindowAdapter
    {
        public void windowClosing (WindowEvent e)
        {
            JFrame f = (JFrame) e.getWindow();
            f.setTitle("Closing window...");
            for (int i = 0; i < 500000000; i++);
            f.setVisible(false);
            f.dispose();
        }
    }

}
```

Components Effecting The State Of Other Components: Class ButtonListener

```
public class MyButtonListener implements ActionListener
{
    public static final int DELAY_TIME = 999999999;
    public void actionPerformed (ActionEvent e)
    {
      JButton aButton = (JButton) e.getSource();
      String s = e.getActionCommand();
      MyFrame aFrame = (MyFrame) aButton.getRootPane().getParent();
```

Components Effecting The State Of Other Components: Class ButtonListener (2)

```
if (s.equals("her"))
{
        JButton himButton = aFrame.getHimButton();
        String title = aFrame.getTitle();
        aFrame.setTitle(himButton.getText());
        himButton.setBackground(Color.green);
        aButton.setBackground(Color.lightGray);
        for (int i = 0; i < DELAY_TIME; i++);
        aFrame.setTitle(title);
}
```

Components Effecting The State Of Other Components: Class ButtonListener (3)

```
  else if (s.equals("him"))
  {

          JButton herButton = aFrame.getHerButton();
          String title = aFrame.getTitle();
          aFrame.setTitle(herButton.getText());
          herButton.setBackground(Color.green);
          aButton.setBackground(Color.lightGray);
          for (int i = 0; i < DELAY_TIME; i++);
          aFrame.setTitle(title);
  }
  else
  {
      :       :
  }
 }
}
```

The JList Class

- Used to provide a graphical and interactive control for a list.
  - This example will create a list of strings but the data is quite flexible.
- Scrollbars are NOT automatically included (you need to add a JList reference to the constructor of a class that provides scrolling capabilities).
- For the complete class refer to the url:
  - *http://java.sun.com/javase/7/docs/api/*
- For online tutorials (Sun):
  - http://download.oracle.com/javase/tutorial/uiswing/components/list.html

Some Important Parts Of The List Class

```
class JList
{
  // The data for the list is stored by another class (some form of ListModel).
  // Returns a reference to the model (needed to add/remove elements).
  ListModel getModel ()

  // Creates a list whose data is an array of objects.
  public ListModel (Object [] listData)

  // Creates an empty list with the data managed by specified model
  public ListModel (ListModel dataModel1)

  // Passes an instance of a class that reacts to list events.
  addListSelectionListener(ListSelectionListener listener)
```

1 The data model is what manipulates the list data. Common options *DefaultListModel*, ListModel

Some Important Parts Of The List Class (2)

```
    // Determines the number of rows that appear..
    setVisibleRowCount (int count)

    // Determine the index of the selected element
    int getSelectedIndex ()

}
```

Adding Scrolling Capability

- As you create an instance of the class that provides scrolling capabilities pass in a reference to the list in the scrolling classes' constructor.
  - E.g., JScrollPane (<*reference to the List*>)
- Then add an instance of this scrolling class to the Frame (or the appropriate container).
  - E.g., aFrame.add (<*reference to the scrolling object*>)

Adding/Removing Elements

- As previously mentioned the ListModel (and not the list) is responsible for manipulating (adding/removing) the list data.
- In order to change the list's membership you need to first get a reference to the list model.
  - E.g., aModel = aList.getModel ()
- Then you can call the appropriate method of the ListModel.
  - Add: aModel.addElement (<*object*>)
  - Remove: aModel.removeElementAt (<*index or object*>)

An Example Employing A List

The complete code for this example can be found in UNIX under the path: /home/233/examples/gui/eighth_JList

An Example Employing A List: The Driver Class

```
public class Driver
{
    public static final int WIDTH = 400;
    public static final int HEIGHT = 300;
    public static void main (String [] args)
    {
        MyFrame aFrame = new MyFrame ();
        aFrame.setSize(WIDTH,HEIGHT);
        aFrame.setVisible(true);
    }
}
```

An Example Employing A List: Class MyFrame

```
public class MyFrame extends JFrame
{
   private JLabel listLabel;
   private JLabel textLabel;
   private JList aList;
   private GridBagLayout aLayout;
   private GridBagConstraints aConstraint;
   private JTextField aTextField;

   public static final int MAX_VISIBLE_ROWS = 3;
   public static final int MAX_ELEMENTS = 10;
```

An Example Employing A List: Class MyFrame (2)

```
public MyFrame ()
{
   MyWindowListener aWindowListener = new MyWindowListener ();
   Object anObject;
   String aString;
   DefaultListModel aModel;
   int size;

   addWindowListener(aWindowListener);
   aConstraint = new GridBagConstraints();
   aList = new JList(new DefaultListModel ());
   initializeList();
   aList.setSelectedIndex(0);
   aTextField = new JTextField ();
   aModel = (DefaultListModel) aList.getModel();
   size = aModel.getSize();
   anObject = aModel.getElementAt(0);
```

An Example Employing A List: Class MyFrame (3)

```
   if (anObject instanceof String)
   {
       aString = (String) anObject;
       aTextField.setText(aString);
   }
   aList.setVisibleRowCount(MAX_VISIBLE_ROWS);
   aList.addListSelectionListener(new MyListListener());
   listLabel = new JLabel(Integer.toString(size));
   textLabel = new JLabel ("Currently selected item");

   aLayout = new GridBagLayout();
   setLayout(aLayout);     // Calling method of super class.
   addWidget(listLabel, 0, 0, 1, 1, GridBagConstraints.NONE);
   addWidget(textLabel, 3, 0, 1, 1, GridBagConstraints.NONE);
   addWidget(new JScrollPane(aList), 0, 1, 2, 3,
                 GridBagConstraints.HORIZONTAL);
   addWidget(aTextField, 3, 1, 1, 1, GridBagConstraints.HORIZONTAL);
 }
```

An Example Employing A List: Class MyFrame (4)

```
public void addWidget (Component widget, int x, int y, int w, int h, int fill)
{
   aConstraint.gridx = x;
   aConstraint.gridy = y;
   aConstraint.gridwidth = w;
   aConstraint.gridheight = h;
   aConstraint.fill = fill;
   aLayout.setConstraints (widget, aConstraint);
   add(widget);        // Calling method of super class.
}
```

An Example Employing A List: Class MyFrame (5)

```
public void initializeList ()
{
   int i;
   DefaultListModel aModel = (DefaultListModel) aList.getModel ();
   for (i = 1; i <= MAX_ELEMENTS; i++)
     aModel.addElement(new String(Integer.toString(i * 10)));
}

public JTextField getTextField ()
{
   return aTextField;
}
```

An Example Employing A List: Private Inner Class (MyListListener)

```
private class MyListListener implements ListSelectionListener
{
    public void valueChanged(ListSelectionEvent e) {
        JList aList = (JList)e.getSource();
        int index = aList.getSelectedIndex();
        DefaultListModel aModel = (DefaultListModel) aList.getModel();
        Object anObject = aModel.getElementAt(index);
        if (anObject instanceof String) {
           String aString = (String) anObject;
           MyFrame aFrame = (MyFrame) aList.getRootPane().getParent();
           JTextField aTextField = aFrame.getTextField();
           aTextField.setText(aString);
        }
    }
}
```

Capturing TextField Events

The complete code for this example can be found in UNIX under the path: /home/233/examples/gui/ninth_JTextField

Capturing TextFieldEvents: Class MyFrame

```
public class MyFrame extends JFrame
{
   private JLabel instructions;
   private JTextField inputField;
   public MyFrame ()
   {
          setLayout(null);
          instructions = new JLabel("Enter some text below and hit return");
          instructions.setBounds(20,100,200,20);
          inputField = new JTextField();
          inputField.setBounds(20,150,200,20);
```

Capturing TextFieldEvents: Class MyFrame (2)

```
            inputField.addActionListener(new ActionListener() {
            public void actionPerformed (ActionEvent e) {
            JTextField aTextField = (JTextField) e.getSource ();
             MyFrame aFrame = (MyFrame)
                                   aTextField.getRootPane().getParent ();
             aFrame.setTitle (aTextField.getText());
          }
         });
          add(instructions);
          add(inputField);
          setVisible(true);
   }
```

References

Books:

- "*Java Swing*" by Robert Eckstein, Marc Loy and Dave Wood (O'Reilly)
- "*Absolute Java*" (4th Edition) by Walter Savitch (Pearson)
- "*Java: How to Program*" (6th Edition) by H.M. Deitel and P.J. Deitel (Pearson)

•Websites:

- Java API specifications: http://download.oracle.com/javase/7/docs/api/
- Java tutorials: http://download.oracle.com/javase/tutorial/uiswing/

You Should Now Know

•The difference between traditional and event driven software

•How event-driven software works (registering and notifying event listeners)

•How some basic Swing controls work

•How to layout components using layout managers and laying them out manually using a coordinate system