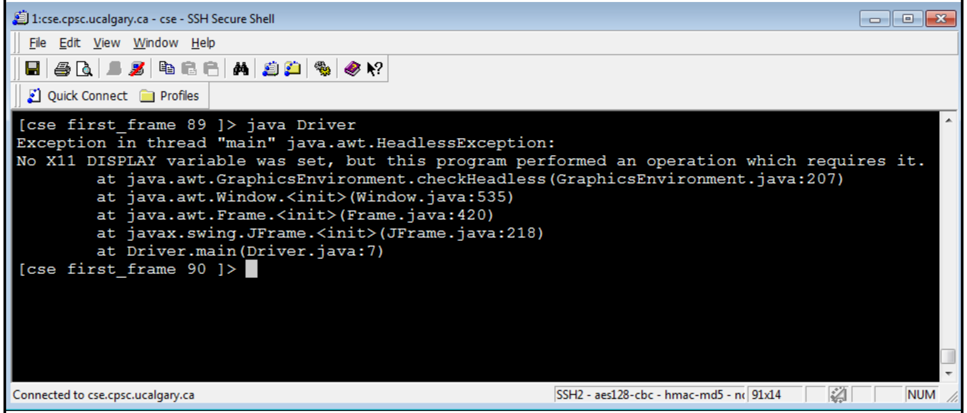
An Introduction To Graphical User Interfaces

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

Don't Run The GUI Code Via SSH!

- The former is graphical
- The latter is text-only


1:cse.cpsc.ucalgary.ca - cse - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles
Connected to cse.cpsc.ucalgary.ca
SSH2 - aes128-cbc - hmac-md5 - nc
91x14
NUM

```
[cse first_frame 89 ]> java Driver
Exception in thread "main" java.awt.HeadlessException:
No X11 DISPLAY variable was set, but this program performed an operation which requires it.
        at java.awt.GraphicsEnvironment.checkHeadless(GraphicsEnvironment.java:207)
        at java.awt.Window.<init>(Window.java:535)
        at java.awt.Frame.<init>(Frame.java:420)
        at javax.swing.JFrame.<init>(JFrame.java:218)
        at Driver.main(Driver.java:7)
[cse first_frame 90 ]>
```

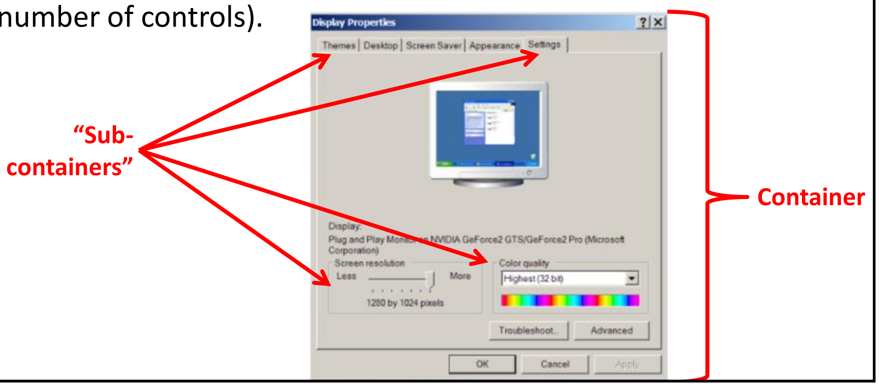
James Tam

Components

- They are many types of graphical controls and displays available:
  - `JButton, JFrame, JLabel, JList, JTextArea, Window`
- A graphical component is also known as a "widget"
- For Sun's online documentation refer to the url:
  - [*http://download.oracle.com/javase/7/docs/api/*](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 other 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).



Display:
Plug and Play Monitor on NVIDIA GeForce2 GTS/GeForce2 Pro (Microsoft Corporation)
Screen resolution: 1280 by 1024 pixels
Color quality: Highest (32 bit)

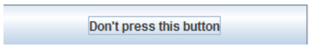
Containers (2)

- You must have at least one container object for your GUI:
  - Examples: JPanel, JWindow, JDialog, JFrame
  - (The most likely one for the assignment is `JFrame`)
- Components which have been added to a container will appear/disappear and be garbage collected along with the container.

James Tam

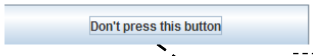
Some Relevant Java GUI libraries

1. Java classes for the `Components` and `Containers`
   - e.g., `JButton` class...
   - ...located in `javax.swing` (`import javax.swing.*` or `import javax.swing.`*`<class name>`*)



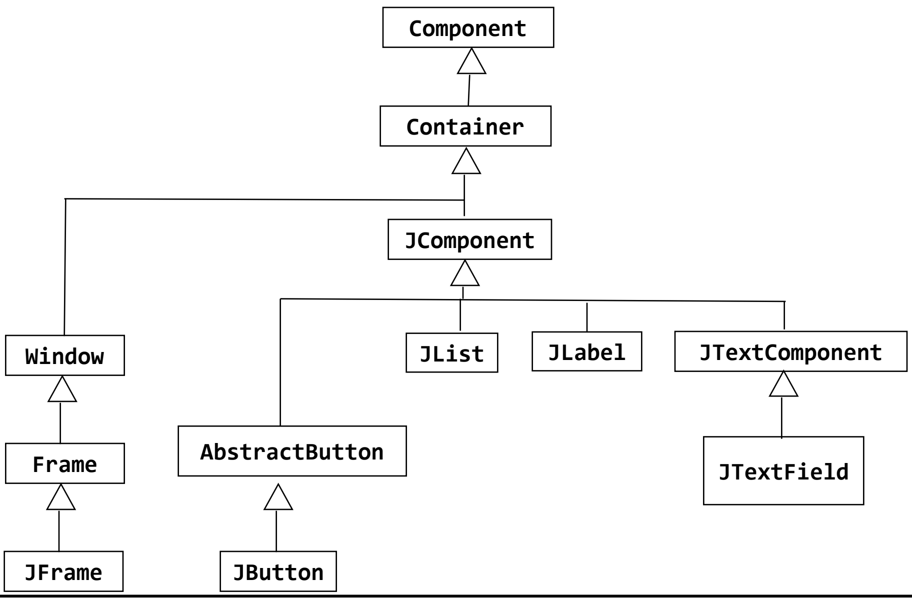
Some Relevant Java GUI libraries (2)

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

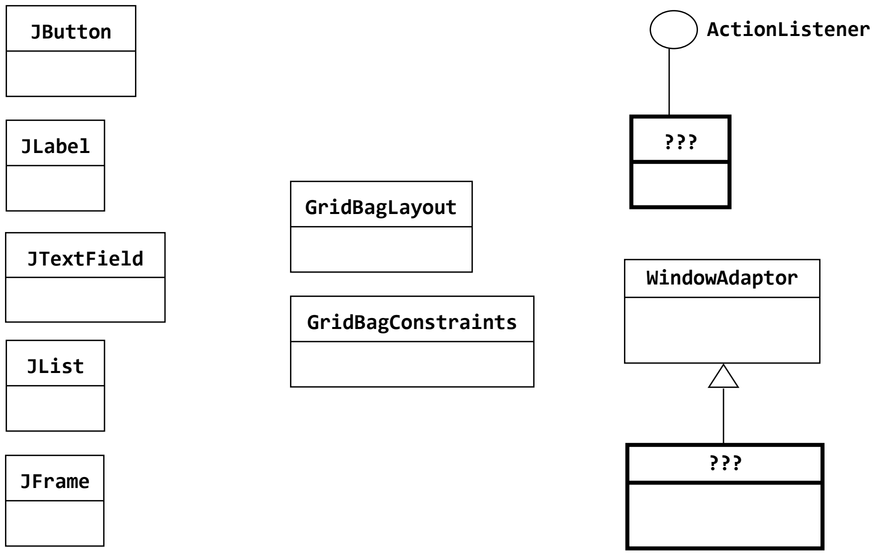


```
class ButtonListener implements ActionListener
{
     public void actionPerformed(ActionEvent e)
    {
         :         :         :
    }
}
```

Hierarchy: Important Widget Classes



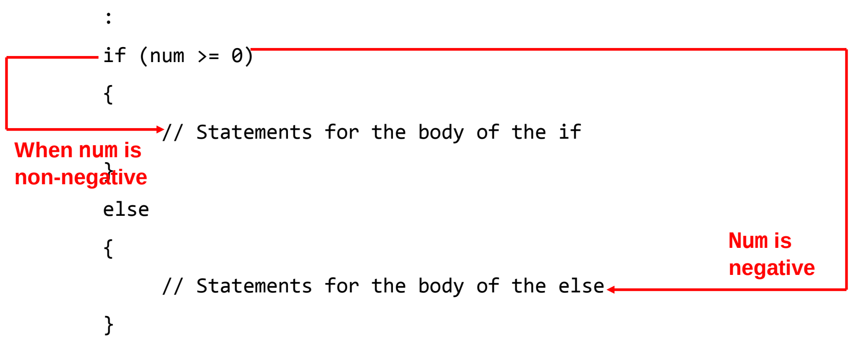
Some Relevant Java GUI Classes For This Section



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
    }
```

Traditional Software

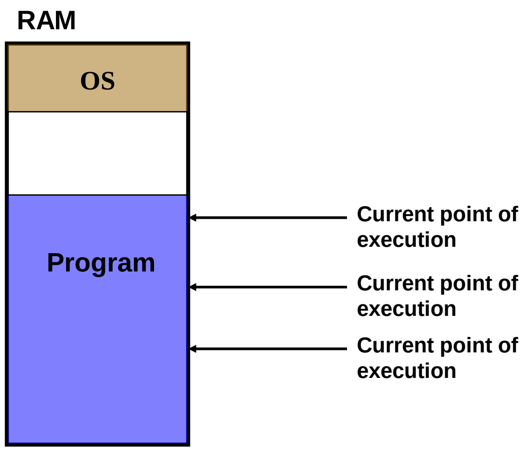
- The user can only interact with the program at places that are specified by the program (e.g., when an input statement is encountered).

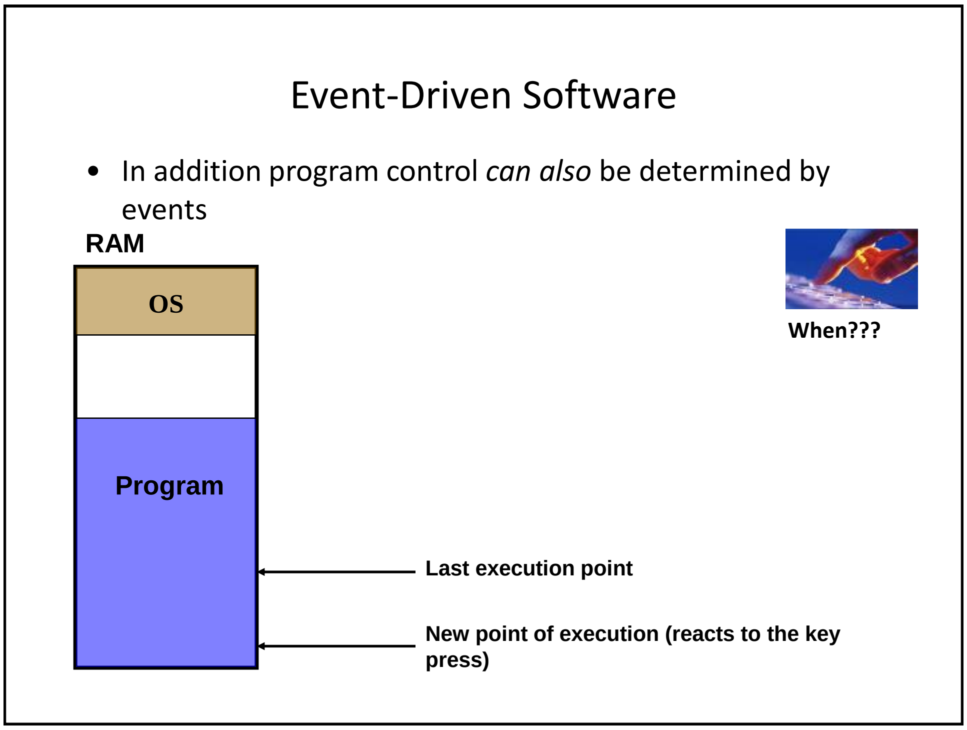
Example

```
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

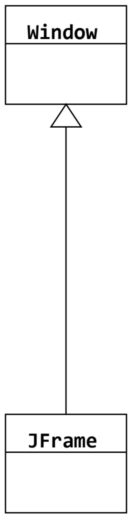
- Program control can be determined by events as well as standard program control statements.
- A typical source of these events is the user.
- These events can occur at any time.

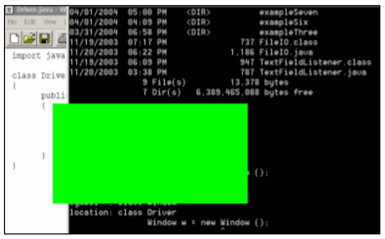
Most Components Can Trigger Events

- 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.

Last name: Tam

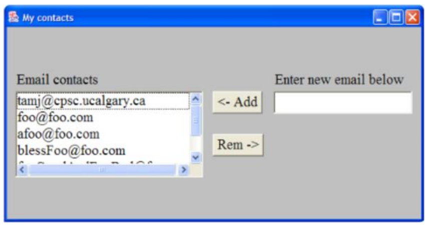
"Window" Classes





```
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
[redacted]
location: class Driver
                Window w = new Window ();
                           ^
```

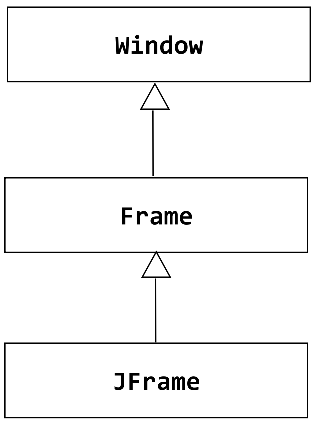


Email contacts

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

Enter new email below

The “Window” Class Hierarchy



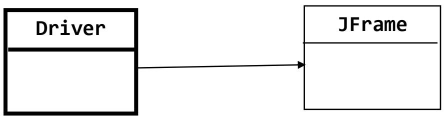
Class `JFrame`

- For full details look at the online API:
  - http://download.oracle.com/javase/7/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)

- Location of the full example:
  `/home/233/examples/gui/1frame`



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

```
import javax.swing.JFrame;
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.

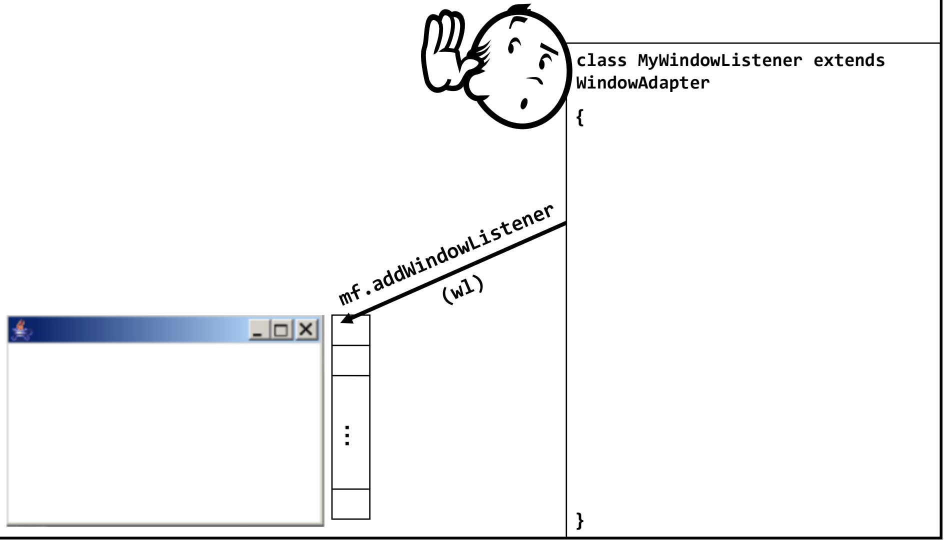
Window Events

- 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

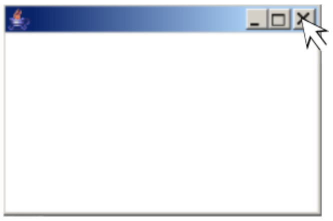
1) The frame must register all interested event listeners.
   – Track where notifications should be sent
2) The user triggers the event by closing the window
3) The window sends a message to all listeners of that event.
   – Send the notifications when the even occurs
4) The window event listener runs the code to handle the event (e.g., save information to a file).
   – When the object with an 'interest' in the event has been notified it executes a method appropriate to react to the event.

1. The Frame Must Register All Interested Event Listeners.

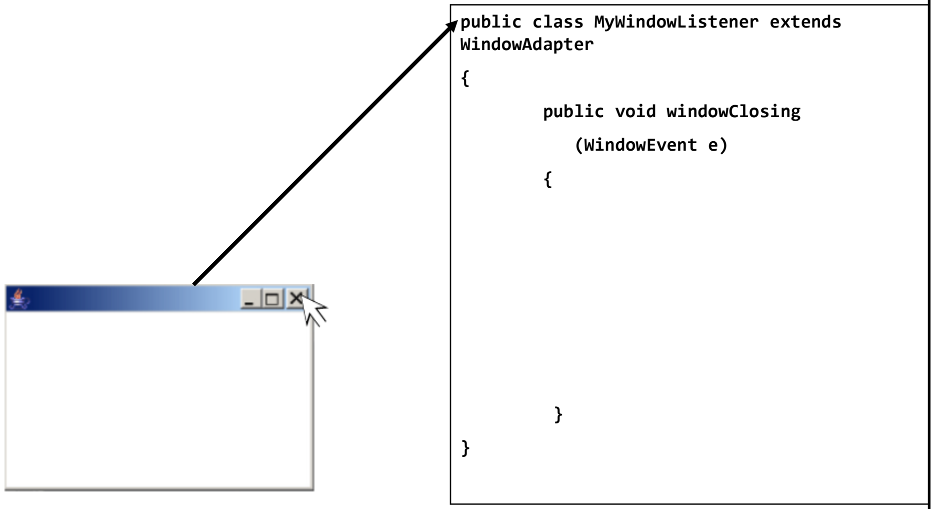


```
class MyWindowListener extends
WindowAdapter
{

}
```

2. The User Triggers The Event By Closing The Window

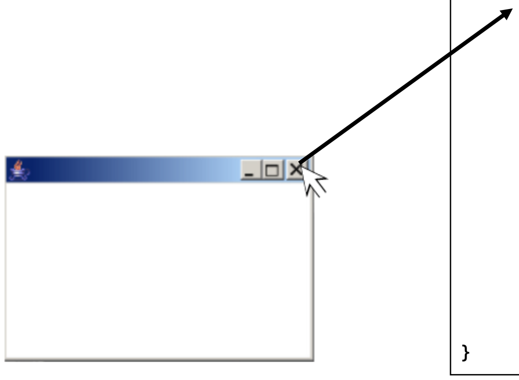


3. The Window Sends A Message To All Listeners Of That Event.



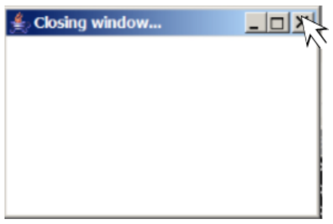
```
public class MyWindowListener extends
WindowAdapter
{
        public void windowClosing
           (WindowEvent e)
        {




         }
}
```

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



```
public class MyWindowListener extends
WindowAdapter
{
        public void windowClosing
          (WindowEvent e)
        {
            /* Code to react to event * /
            JFrame aFrame = (JFrame)
              e.getWindow();
            aFrame.setTitle("Closing
              window...");
            aFrame.setVisible(false);
            aFrame.dispose();
        }
}
```

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

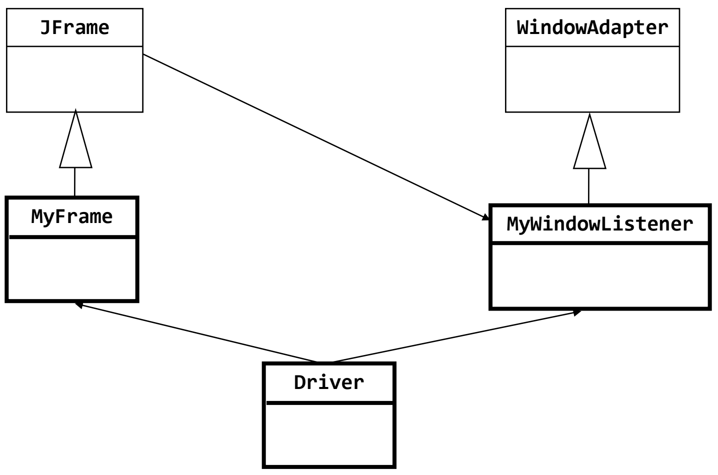


```
public class MyWindowListener extends
WindowAdapter
{
        public void windowClosing
          (WindowEvent e)
        {
            /* Code to react to event * /
            JFrame aFrame = (JFrame)
              e.getWindow();
            aFrame.setTitle("Closing
              window...");
            aFrame.setVisible(false);
            aFrame.dispose();
        }
}
```

An Example Of Handling A Frame Event

- Location of the example:
  `/home/233/examples/gui/2windowEvents`

An Example Of Handling A Frame Event (2)



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.
}
```

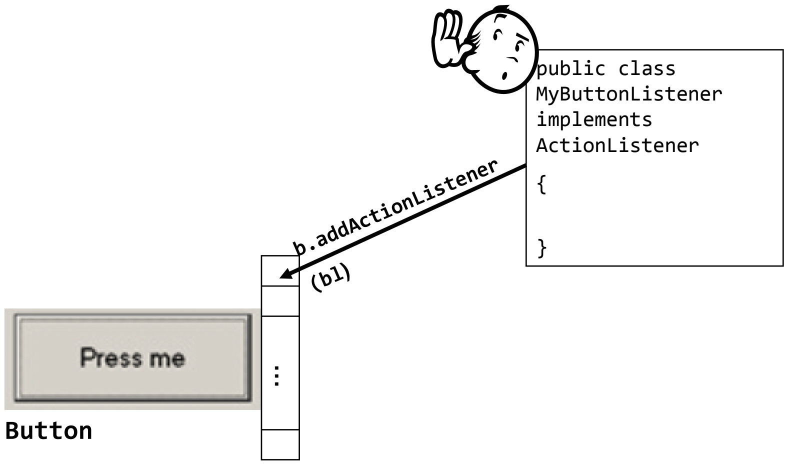
Class MyWindowListener

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

public class MyWindowListener extends WindowAdapter {
           public void windowClosing (WindowEvent e) {
           JFrame aFrame = (JFrame) e.getWindow();
           aFrame.setTitle("Closing window...");
           // Pause program so user can see the window text
           try
               Thread.sleep(3000);
           catch (InterruptedException ex)
               System.out.println("Pausing of program was
                 interrupted");
           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

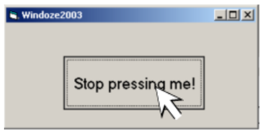
3. The Component Sends A Message To All Registered Listeners For That Event

```
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!");
    }
}
```

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

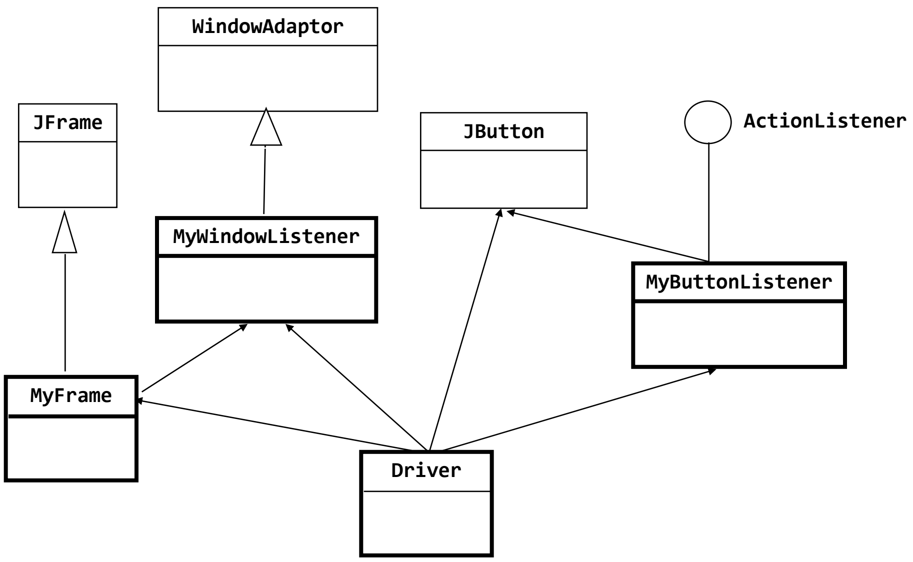


```
public class MyButtonListener
implements ActionListener
{
    public void actionPerformed
      (ActionEvent e)
    {
        JButton b = (JButton)
          e.getSource();
        b.setLabel("Stop pressing
         me!");
    }
}
```

An Example Of Handling A Button Event

- Location of the example:
  `/home/233/examples/gui/3ButtonEvents`

An Example Of Handling A Button Event (2)



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!");
    }
}
```

How To Handle The Layout Of Components

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

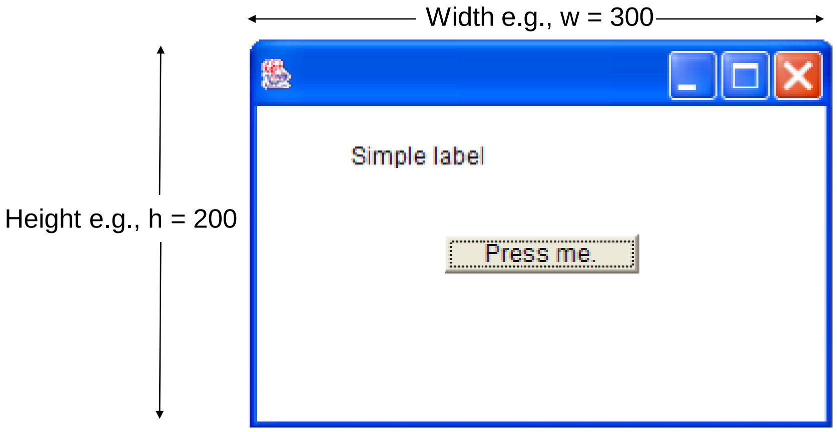
How To Handle The Layout Of Components

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

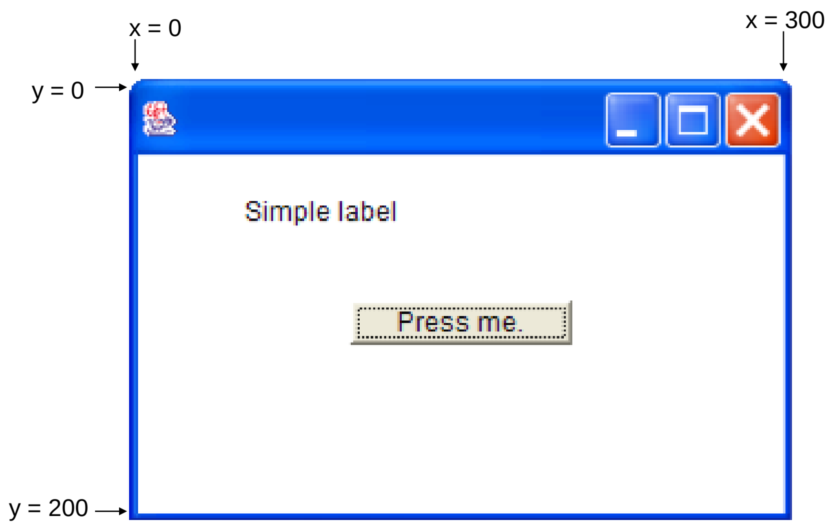
Layout Is Based On Spatial (X,Y) Coordinates

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

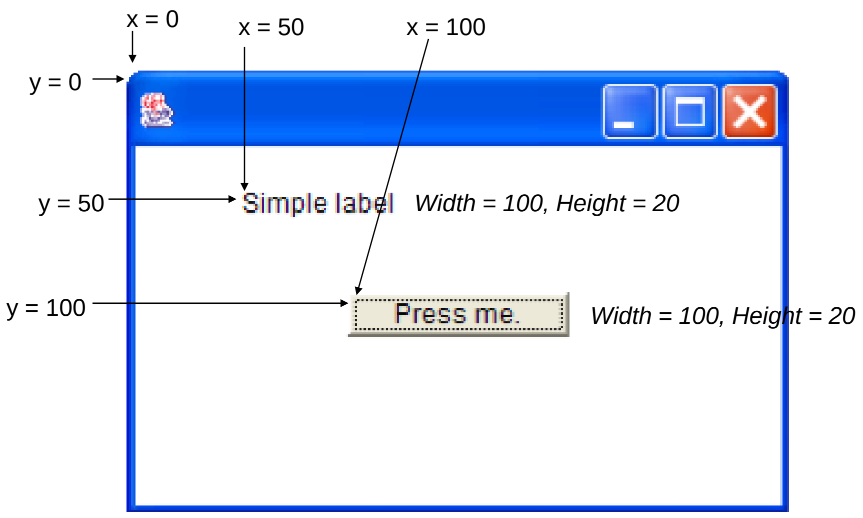
```
my.setSize(300,200);
```



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 With Manual Layout

- Location of the example:
  `/home/233/examples/gui/4manualLayout`

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

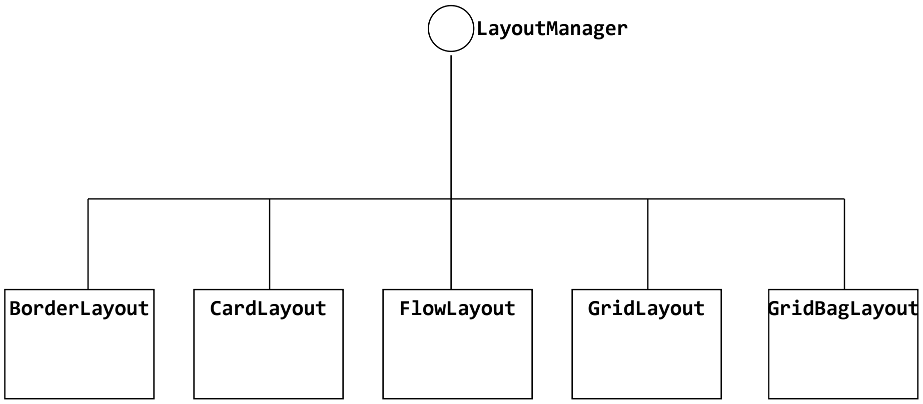
```
    public static void main (String [] args) {
        JFrame aFrame = new JFrame ();
        aFrame.setLayout(null);
        aFrame.setSize (WIDTH_FRAME,HEIGHT_FRAME);
        JButton aButton = new JButton("Press me.");
        aButton.setBounds(X_COORD_BUTTON,
                          Y_COORD_BUTTON,
                          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

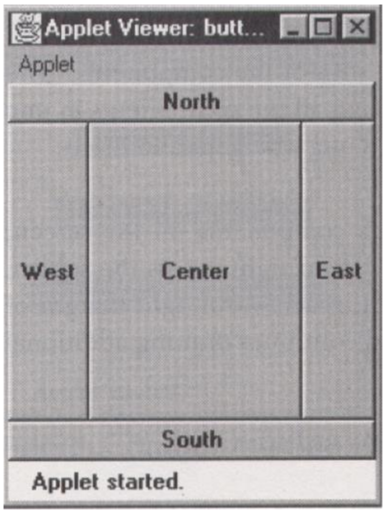
1. Manually set the coordinates yourself
2. **Use one of Java's built-in layout manager classes**

Java Layout Classes

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

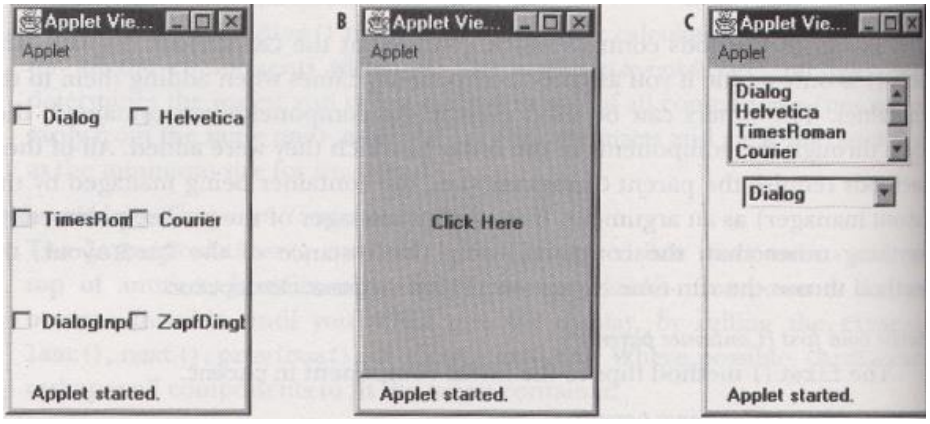


BorderLayout ("Compass Directions")



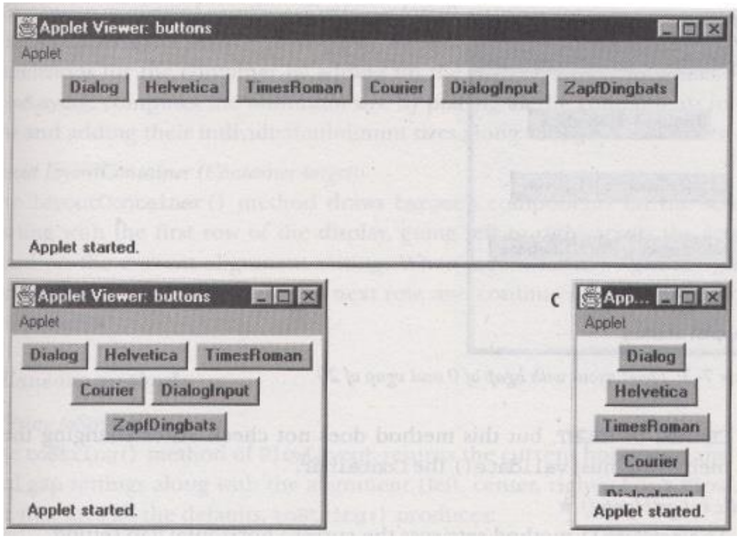
From Java: AWT Reference p. 256

`CardLayout` (“Tab-Like”)



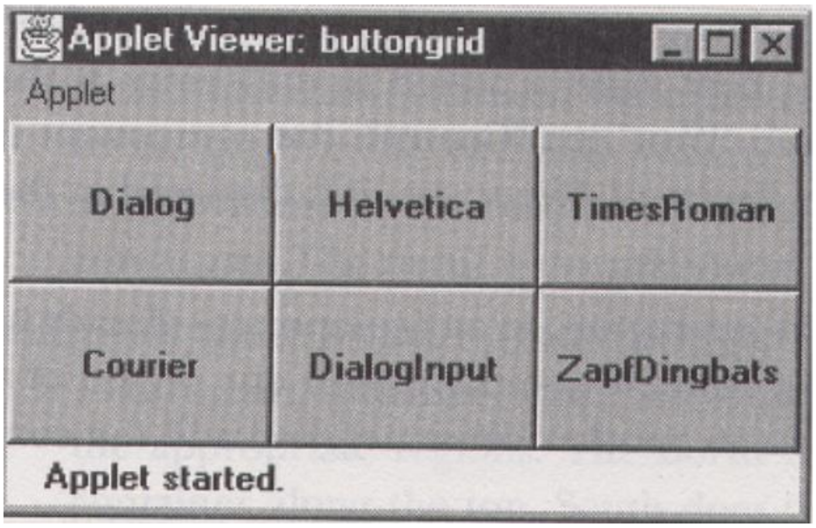
From Java: AWT Reference p. 264

`FlowLayout` (Adapts To Resizing “Web-Like”)



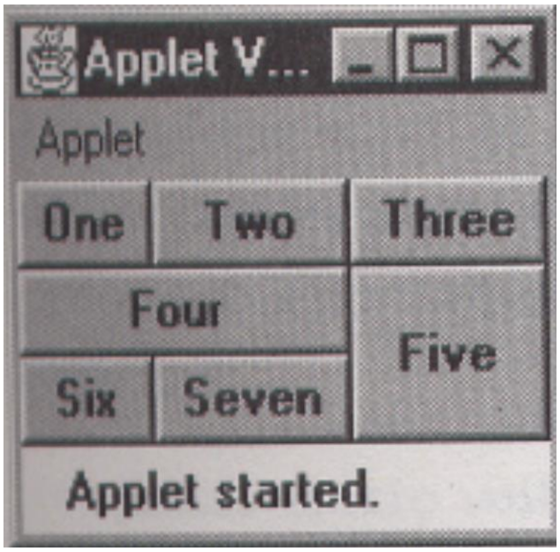
From Java: AWT Reference p. 253

GridLayout



From Java: AWT Reference p. 260

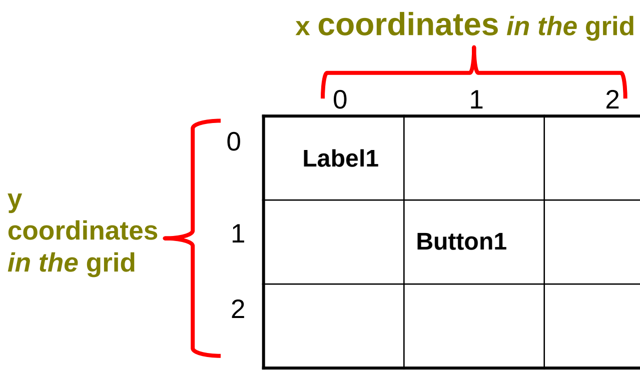
GridBagLayout



From Java: AWT Reference **p. 269**

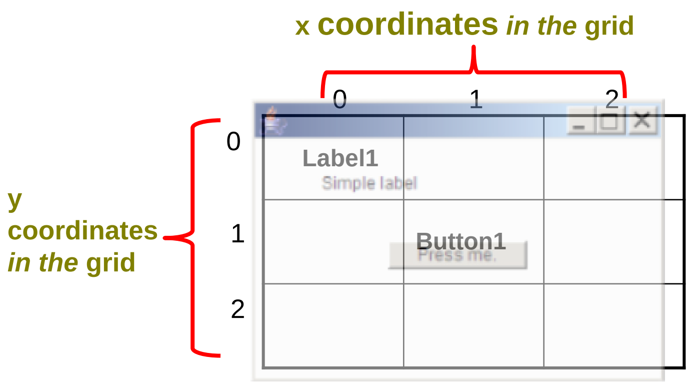
Implementing A GUI When Using The `GridBagLayout`

- Use graph paper or draw out a table.



Implementing A GUI When Using The `GridBagLayout`

- Use graph paper or draw out a table.



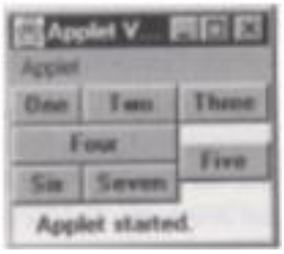
GridBagConstraints

- 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 for a particular `GridBagLayout`.
- For more complete information see:
  - *http://java.sun.com/javase/7/docs/api/java/awt/GridBagConstraints.html*

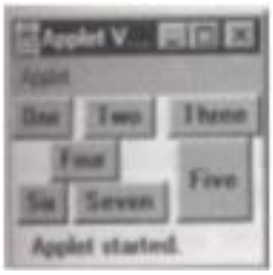
Some Important Parts Of The GridBagConstraints Class

```
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;
```

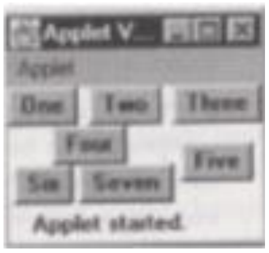
`GridBagContraints`: Fill Values



Horizontal



Vertical



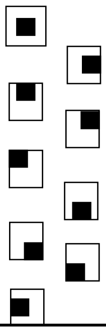
None

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;
```

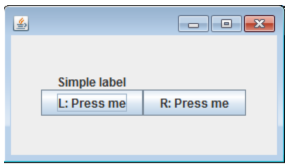
Some Important Parts Of The GridBagConstraints Class (3)

```
// 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;

// Only if the component is smaller than the available space.
// Determine the anchor direction
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;
```

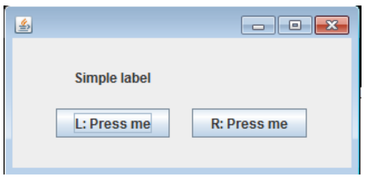


Some Important Parts Of The GridBagConstraints Class (4)

```
// With a particular 'cell' in the grid this attribute
// specifies the amount of padding around the component
// to separate it from other components.
// Usage:
// insets = new Insets(<top>,<left>,<bottom>,<right>);
// Example (Set top, left, bottom, and right)
// insets = new Insets(0, 0, 0, 0); // No padding (default)
public insets;
```



Insets = 0: no padding



Insets = 10: many spaces/padding

An Example Using The GridBagLayout

•Location of the example:
/home/233/examples/gui/5gridbaglayout

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 left;
    private JButton right;
    private JLabel aLabel;
    private GridBagLayout aLayout;
    GridBagConstraints aConstraint;

    public MyFrame () {
        MyWindowListener aWindowListener = new MyWindowListener ();
        addWindowListener(aWindowListener);
        aConstraint = new GridBagConstraints();
        Scanner in = new Scanner(System.in);
        System.out.print("Buffer size to pad the grid: ");
        int padding = in.nextInt();
```

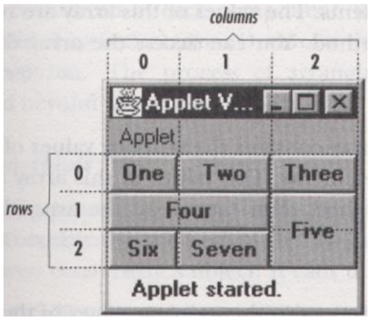
An Example Using The GridBagLayout: Class MyFrame (2)

```
        left = new JButton("L: Press me");
        right = new JButton("R: Press me");
        MyButtonListener aButtonListener = new MyButtonListener();
        left.addActionListener (aButtonListener);
        right.addActionListener (aButtonListener);
        aLabel = new JLabel("Simple label");
        aConstraint.insets = new
           Insets(padding,padding,padding,padding);
        aLayout = new GridBagLayout();
        setLayout(aLayout);     // Calling method of super class.
        addWidget(aLabel, 0, 0, 1, 1);
        addWidget(left, 0, 1, 1, 1);
        addWidget(right, 1, 1, 1, 1);
    }
```

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.
    }
} // End of definition for class MyFrame
```

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

Layout Of GUI Components

- JT's note (and opinion): learning how to layout GUI components manually will teach you "how things work".
  - That's because you have to handle many details yourself (either manually or by using a layout class).
  - Except when writing small programs with a simple GUI (assignment) doing things manually is just too much of a hassle.
    - The programmer focuses on the wrong details (how do I get the programming language to 'do stuff' as opposed to how do I create a GUI that is 'user-friendly').
  - In other cases ('real life programs') an IDE is used.
  - Some examples:
    - Sun's NetBeans IDE: http://docs.oracle.com/javase/tutorial/uiswing/learn/index.html
    - IBM's Eclipse IDE: http://www.ibm.com/developerworks/opensource/library/os-ecvisual/

Components Effecting The State Of Other Components

- Location of the example:

`/home/233/examples/gui/6controlAffectControls`

Components Effecting The State Of Other Components: The `Driver` Class

```
public class Driver
{
    public static final int WIDTH = 800;
    public static final int HEIGHT = 600;
    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 aLabel1;
   private JLabel aLabel2;
   private JButton aButton;
   private MyButtonListener aButtonListener;
```

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

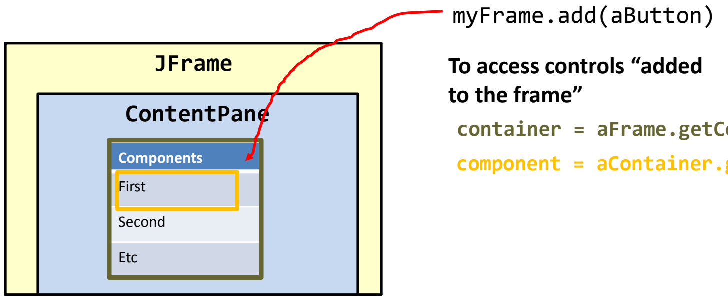
```
public MyFrame ()
{
      MyWindowListener aWindowListener =
        new MyWindowListener ();
      addWindowListener(aWindowListener);
      aLabel1 = new JLabel("Label 1");
      aLabel2 = new JLabel("Label 2");
      aLabel1.setBounds(100,100,100,30);
      aLabel2.setBounds(300,100,100,30);
```

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

```
      aLabel1 = new JLabel("Label 1");
      aLabel2 = new JLabel("Label 2");
      aLabel1.setBounds(100,100,100,30);
      aLabel2.setBounds(300,100,100,30);
      aButtonListener = new MyButtonListener();
      aButton = new JButton("Press for multiple effects");
      aButton.addActionListener(aButtonListener);
      aButton.setBounds(150,300,200,50);
      add(aLabel1);
      add(aLabel2);
      add(aButton);
      setLayout(null);
    }
    public JLabel getLabel1 () { return aLabel1; }
    public JLabel getLabel2 () { return aLabel2; }
}
```

Note: JFrame Containment

- A JFrame actually contains just one GUI component, the content pane.
- GUI widgets that appear to be added to the JFrame are actually added to the content pane (a container in and of itself). Get the components inside the content pane to actually get the widgets that appeared to be added to the JFrame.



To access controls "added to the frame"

```
container = aFrame.getContentPane()
component = aContainer.getComponent(0)
```

James Tam

Components Effecting The State Of Other Components: Class MyButtonListener

```
public void actionPerformed (ActionEvent e)
{
        JButton aButton = (JButton) e.getSource();
        MyFrame aFrame = (MyFrame)
          aButton.getRootPane().getParent();
        JLabel aLabel1 = aFrame.getLabel1();
        JLabel aLabel2 = aFrame.getLabel2();

        Container aContainer = aFrame.getContentPane();
        // First item added to list
        Component aComponent = aContainer.getComponent(0);
        if (aComponent instanceof JLabel) {
            aLabel1 = (JLabel) aComponent;
            aLabel1.setText("Effect1");
        }
```

James Tam

Components Effecting The State Of Other Components: Class `MyButtonListener` (2)

```
        // Second item added to list
        aComponent = aContainer.getComponent(1);
        if (aComponent instanceof JLabel) {
            aLabel2 = (JLabel) aComponent;
            aLabel2.setText("Effect1");
        }
}
```

James Tam

Last Example: Critique

- There was one method handles events for all the buttons.
- Inside that method there was a need to ‘identify’ the source of the event.
  - The method could get very long even though there are few sources of events (buttons)
  - What if the GUI has dozens of buttons!

```
public void actionPerformed (ActionEvent e)
{
   String s = e.getActionCommand();
   if (s.equals(“button1")) {

   }
   if (s.equals(“button2")) {

   }
}
```

Anonymous Objects/Anonymous 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.
- Creating an anonymous object:

No reference name

One advantage: code for widget and event handler are in the same place.

Awkward if complex programming is required.

```
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!");
                           }
```

An Example Using Anonymous Class And Object

- Location of the example:

`/home/233/examples/gui/7controlAffectControlsAnonymousObjectClass`

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.setTitle("Original");
         aFrame.setSize(WIDTH,HEIGHT);
         aFrame.setVisible(true);
     }
}
```

Class MyFrame

```
public class MyFrame extends Jframe
{
    private JLabel aLabel;
    private GridBagLayout aLayout;
    private GridBagConstraints aConstraint;
    private JButton left;
    private JButton right;
    public MyFrame ()
```

Class MyFrame (2)

```
public MyFrame () {
    MyWindowListener aWindowListener =
      new MyWindowListener ();
    addWindowListener(aWindowListener);
    aConstraint = new GridBagConstraints();

   left = new JButton("LEFT: Press right button.");
   left.setBackground(Color.lightGray);
```

Class MyFrame (3)

```
   left.addActionListener(new ActionListener()
   {  // class definition
       public void actionPerformed(ActionEvent e)  {
           // method definition: left button
          JButton left = (JButton) e.getSource();
          MyFrame aFrame = (MyFrame)
            left.getRootPane().getParent();
          String title = aFrame.getTitle();
          aFrame.setTitle("Left pressed");
          right.setBackground(Color.green);
          left.setBackground(Color.lightGray);
          timeDelay();
         aFrame.setTitle(title);
       } // End method definition
    } // End class definition
  ); // End of parameter list for addActionListener()
```

James Tam

Class MyFrame (4)

```
right = new JButton("RIGHT: Press left button");
right.setBackground(Color.lightGray);
right.addActionListener(new ActionListener()
{  // Class definition
    public void actionPerformed(ActionEvent e)  {
        // Method definition
        JButton right = (JButton) e.getSource();
        MyFrame aFrame = (MyFrame)
          right.getRootPane().getParent();
        String title = aFrame.getTitle();
        JButton left = aFrame.getLeft();
        aFrame.setTitle("Right pressed");
        left.setBackground(Color.green);
        right.setBackground(Color.lightGray);
        timeDelay();
        aFrame.setTitle(title);
    }
```

Class MyFrame (5)

```
    private void timeDelay ()
    {
        try {
               Thread.sleep(3000);
        }
        catch (InterruptedException e) {
               System.out.println("Problem with pasuing of the
                 program");
        }
    }
    public JButton getLeft() { return(left); }
    public JButton getRight() { return(right); }
}
```

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 (inner class hidden from all classes except the outer class).
- Similar to declaring anonymous objects, nesting classes may be used when creating event listeners.

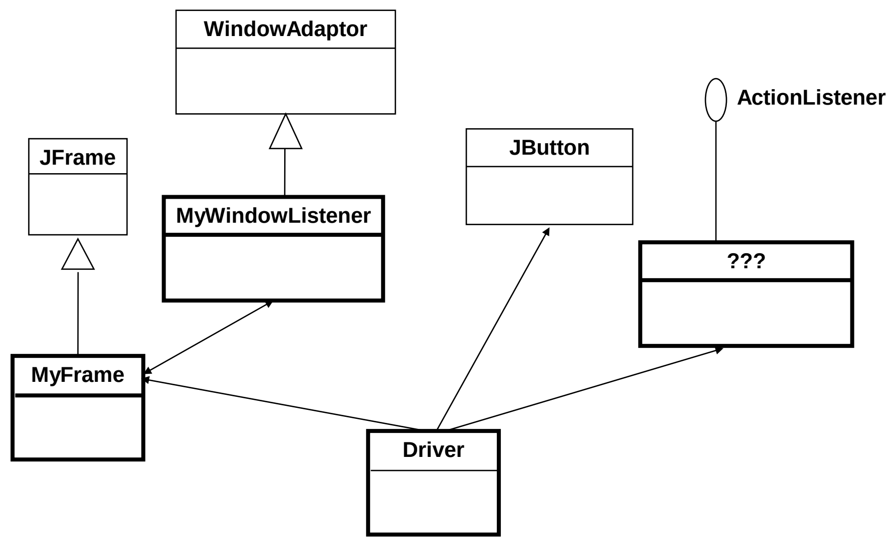
1 For more information: http://download.oracle.com/javase/tutorial/java/javaOO/nested.html

Example: Inner Classes

•Location Of example:

`/home/233/examples/gui/8buttonAlternateInner`

Example: Inner Classes (2)



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!");
                     } // End: Defining method actionPerformed
               }  // End: Defining anonymous object/class
            );  // End: Parameter list for addActionListener

            aFrame.add(aButton);
            aFrame.setVisible(true);
      }
}
```

Class MyFrame: Outline

```
public class MyFrame extends JFrame
{
      // MyFrame's private parts
       public MyFrame ()
      {
            :    :
      }
      // Inner class defined within the MyFrame class.
      //  Private because it's only used by the MyFrame class.
      private class MyWindowListener extends WindowAdapter
      {
          public void windowClosing (WindowEvent e)
          {
               :     :
          }
      }
}
```

NOTE: The inner class can access the outer class' privates! "Friend"

Definition of class MyWindowListener entirely within definition of class MyFrame

- Listens for events for that window

Class MyFrame (2)

```
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 (3)

```
      // Inner class defined within the MyFrame class.
      // Private because it's only used by the MyFrame class.
      private class MyWindowListener extends WindowAdapter {
        public void windowClosing (WindowEvent e) {
           JFrame aFrame = (JFrame) e.getWindow();
           aFrame.setTitle("Closing window...");
           delay();
           aFrame.setVisible(false);
           aFrame.dispose();
        }
      }  // End: Definition of class MyWindowListener

      private void delay() {
           try  {
              Thread.sleep(3000); }
           catch (InterruptedException ex)  {
              System.out.println("Pausing of program was interrupted");
  }
        }
} // End: Definition of class MyFrame
```

Proof that the inner class can access the outer class' privates

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/
  - Java tutorial (layout): http://docs.oracle.com/javase/tutorial/uiswing/layout/using.html

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
  - Capturing common events for the controls such as a button press
- How to layout components using layout managers and laying them out manually using a coordinate system
- How to use/why use anonymous and inner classes