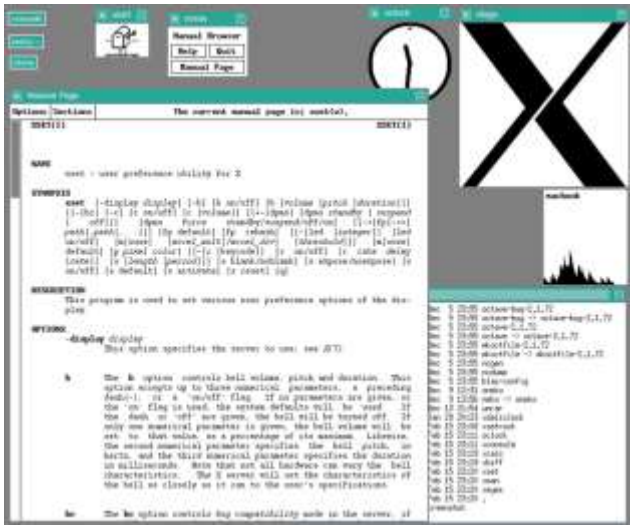
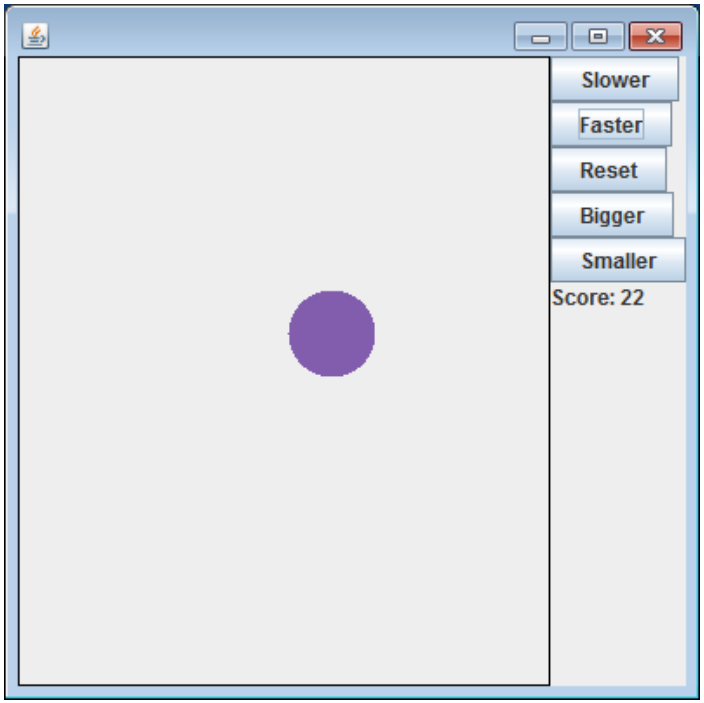
Graphical User Interfaces





Overview

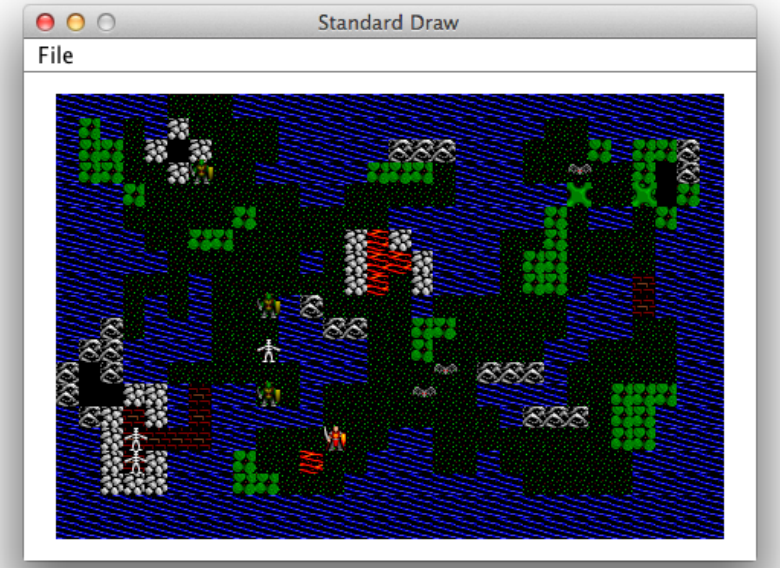
- Command line versus GUI apps
- Java GUI applications
  - JFrame
  - Layout managers
  - Popular widgets
- Events
  - Action listeners
  - Inner listener classes
- Drawing things (without StdDraw)
  - JPanel

User interfaces

- Command Line Interface (CLI)
  - Good for experts
  - Automating thing via scripts (e.g. DOS batch files)
  - Must remember command line options
  - Easy to use remotely (e.g. over SSH)
  - Low resource consumption
- Graphical User Interface (GUI)
  - Good for novices
  - Difficult to automate (e.g. macros in Office)
  - Good for presenting multiple views, graphical data
  - Remote access more difficult (e.g. VNC, Remote Desktop)
  - High resource consumption

Java GUIs

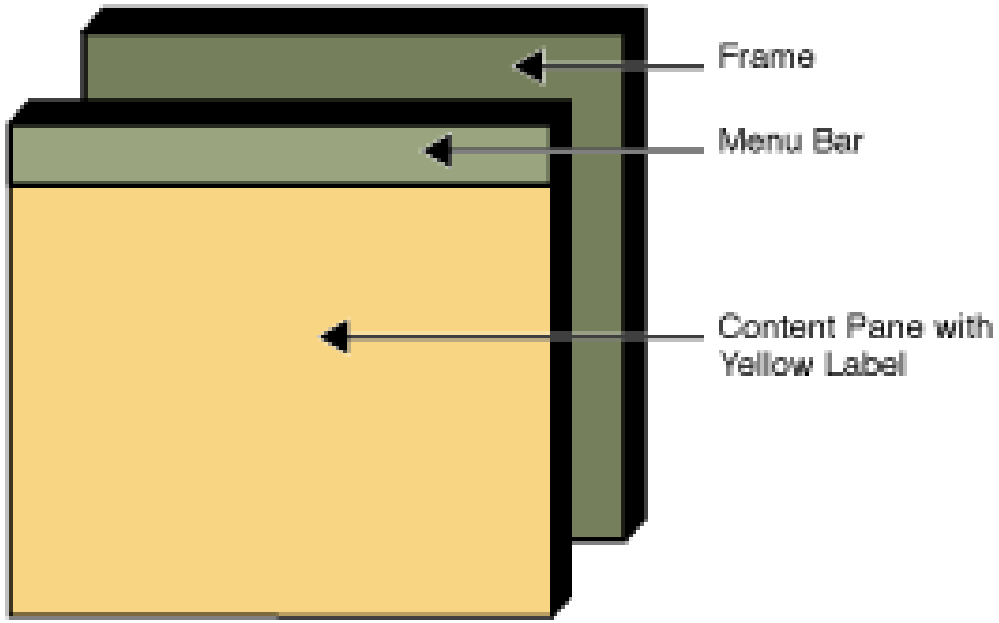
- Thus far: StdDraw
  - Good for drawing
  - But no GUI widgets:
    - Buttons
    - Combo boxes
    - Text fields
    - Dialog boxes



- Today: doing it ourselves
  - Use the Java API to create a GUI
  - Creating common widgets
  - Creating an area to draw things on
  - In practice, often done with the help of a GUI builder

Top-level container

- JFrame
  - Container that holds all the GUI elements
  - main() method creates



JFrame example

- GUI with a single giant button
  - Button doesn't do anything (yet)


```
import javax.swing.*;

public class SimpleButton
{
   public static void main(String [] args)
   {
      JFrame frame = new JFrame();
      JButton button = new JButton("click me!");
      frame.getContentPane().add(button);

      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      frame.setSize(300,300);
      frame.setVisible(true);
   }
}
```

Create the frame and a button. Add the button to the content portion of the frame.

Makes the X box close everything.

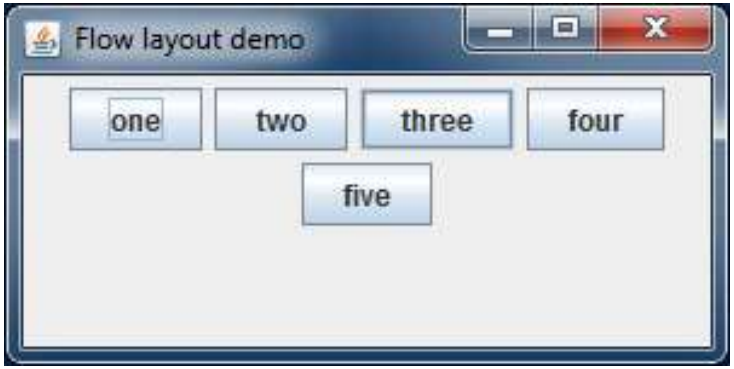
Make a reasonable size and actually display.

Adding multiple things

- Layout manager
  - Handles arranging multiple things in the frame

MultiButtons.java

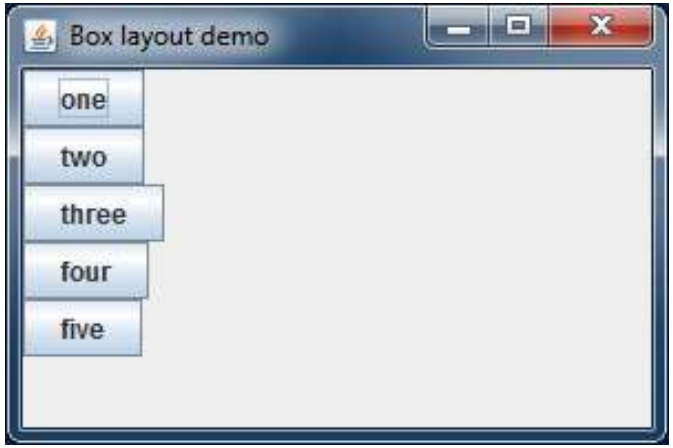
- BorderLayout
  - Default type for a frame
  - 5 regions: NORTH, SOUTH, EAST, WEST, CENTER

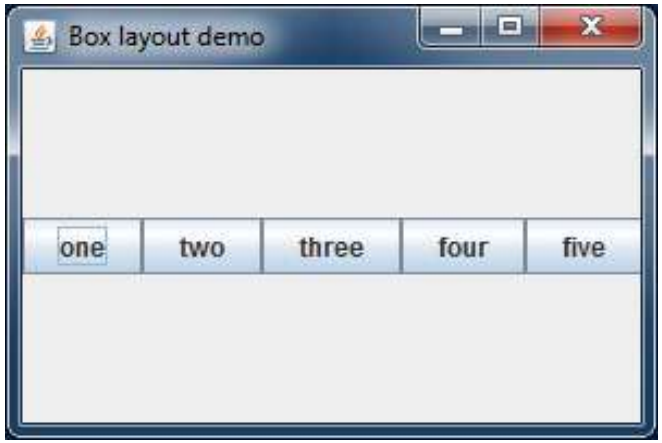


MultiButtonsFlow.java

- FlowLayout
  - Default type for a panel
  - Left to right
  - Create multiple rows if needed

Adding multiple things





MultiButtonsBox.java

- BoxLayout
  - Vertical stack of components
    - BoxLayout.Y_AXIS
  - Horizontal row
    - BoxLayou.X_AXIS
  - Won't rearrange if you resize window

Handy JFrame methods

- Some JFrame methods:

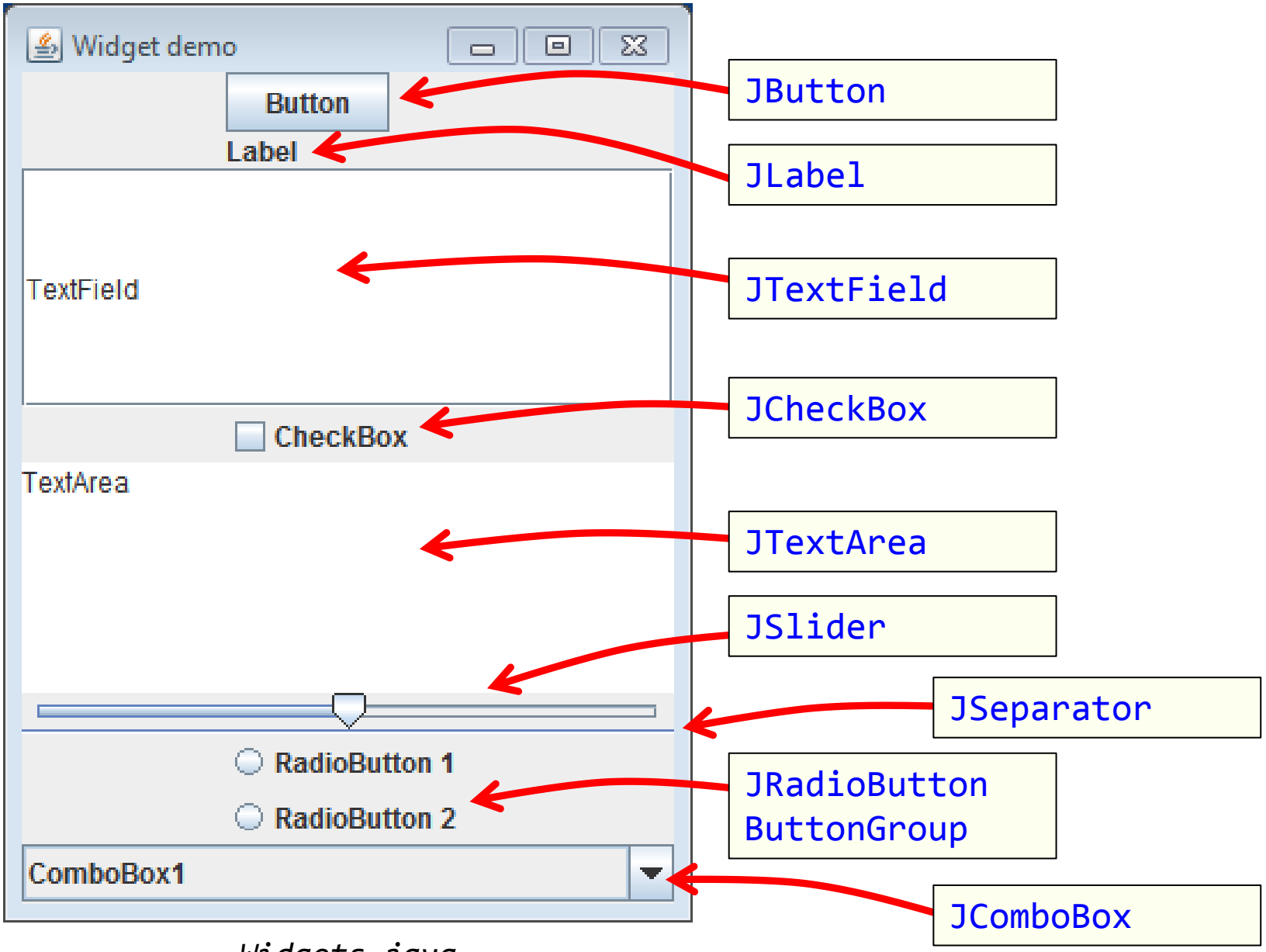
Method	Description
`JFrame()`	Default constructor, a window with no name
`JFrame(String title)`	Constructor, give the window a title
`setLayout(LayoutManager mgr)`	Change the layout manager for the frame
`setDefaultCloseOperation(int op)`	What to do when the frame is closed
`setSize(int width, int height)`	Sets to the given width and height
`setVisible(boolean b)`	Show or hide depending on value of b
`getContentPane()`	Returns the content pane where we add things
`setTitle(String s)`	Change the title of the window

Handy JButton methods

- Some JButton methods:

Method	Description
`JButton(String s)`	Constructor a new button with the given label
`setText(String s)`	Change the button's label
`setEnabled(boolean b)`	Enables or disables the button
`setPreferredSize(Dimension d)`	Tell layout manager how big you would like the button to be

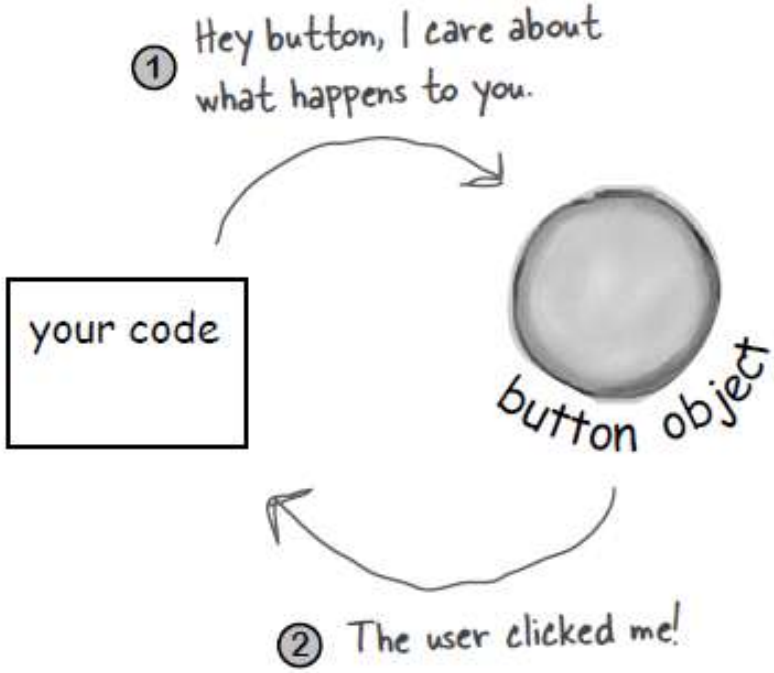
Widget'o'rama



Widgets.java

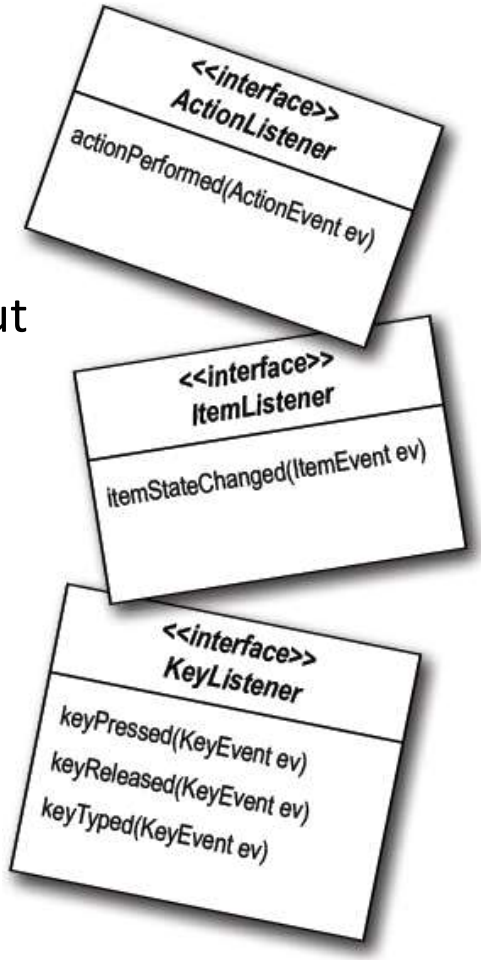
GUI events

- Events
  - Something (usually the user) triggers
  - Examples:
    - The user clicking a button
    - Moving the mouse around
    - Changing the selection in a combo box list



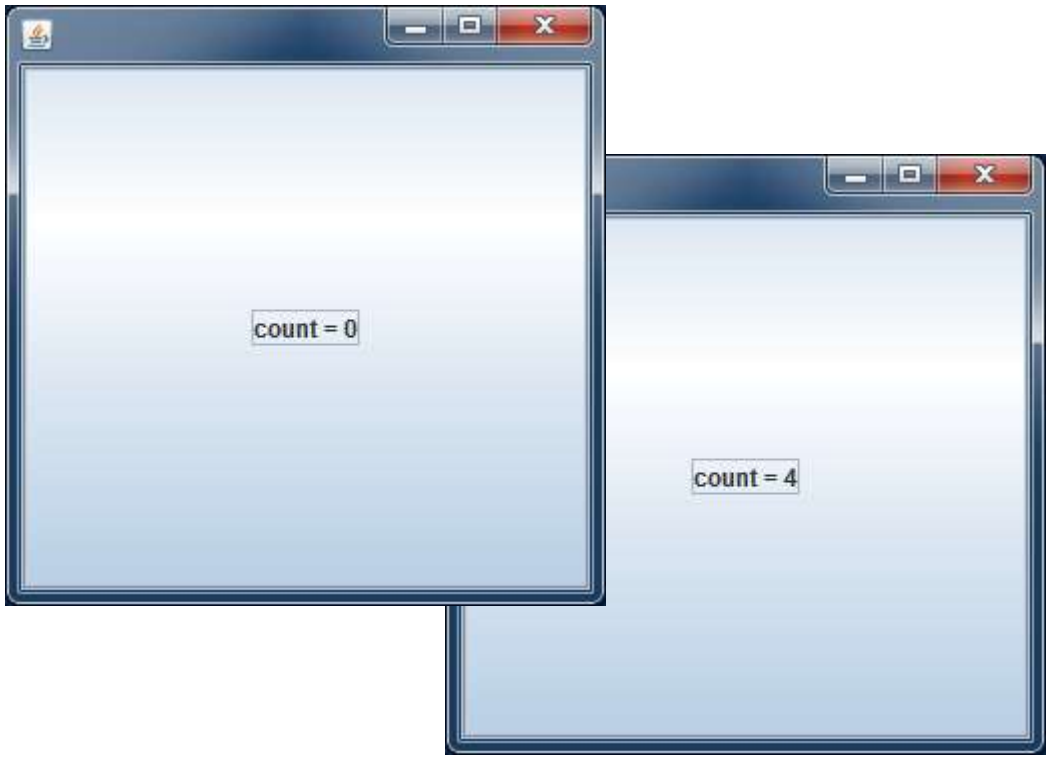
Event listeners and sources

- Event listener
  - Code that is called when an event occurs
  - Done by implementing a Java interface
    - Exactly which one depends on what you care about
  - Registering your listener with the object
- Event source
  - The object that generates the event
  - e.g. the JButton object
  - Normally you'll be handling events
    - Not generating them



Button counter example

- Single button window
  - Every time button is pushed, increment counter
  - Display counter as label of the button



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

public class ButtonCount implements ActionListener
{
   private int count = 0;
   private JButton button;

   public void actionPerformed(ActionEvent event)
   {
      count++;
      button.setText("count = " + count);
   }

   public void go()
   {
      JFrame frame = new JFrame();
      button = new JButton("count = " + count);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.getContentPane().add(button);
      frame.setSize(300,300);
      frame.setVisible(true);
      button.addActionListener(this);
   }

   public static void main(String [] args)
   {
      ButtonCount gui = new ButtonCount();
      gui.go();
   }
}
```

Multiple listeners

- Listening for multiple buttons
  - Single `actionPerformed()` method
    - Use the passed in event object
    - Test which button object triggered the method
  - Multiple inner classes
    - Each classes `implements actionListener`
    - Each has its own `actionPerformed()` method

Single listener approach

```
public class FarmListener implements ActionListener
{
   private JButton buttonCow;
   private JButton buttonDog;
   private JButton buttonCat;

   public void actionPerformed(ActionEvent event)
   {
            if (event.getSource() == buttonCow) StdAudio.play("cow.wav");
      else if (event.getSource() == buttonDog) StdAudio.play("dog.wav");
      else if (event.getSource() == buttonCat) StdAudio.play("cat.wav");
   }

   public void go()
   {
      ...
      buttonCow.addActionListener(this);
      buttonDog.addActionListener(this);
      buttonCat.addActionListener(this);
   }
   ...
}
```

Inner listener approach

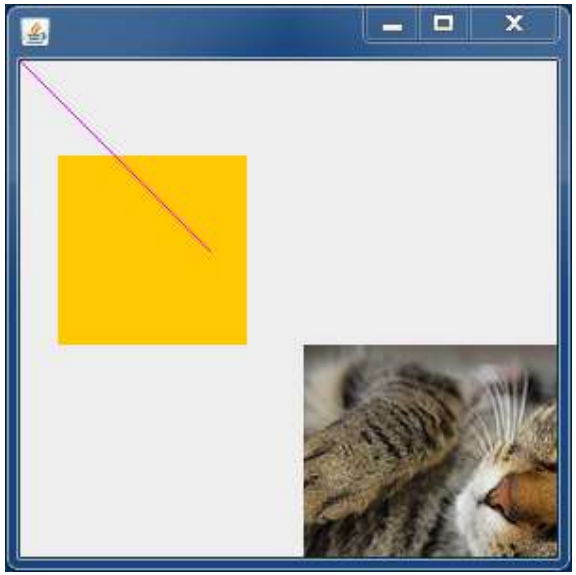
```
public class FarmInner
{
   private JButton buttonCow;
   private JButton buttonDog;
   private JButton buttonCat;

   class CowListener implements ActionListener
   {
      public void actionPerformed(ActionEvent event) { StdAudio.play("cow.wav"); }
   }
   class DogListener implements ActionListener
   {
      public void actionPerformed(ActionEvent event) { StdAudio.play("dog.wav"); }
   }
   class CatListener implements ActionListener
   {
      public void actionPerformed(ActionEvent event) { StdAudio.play("cat.wav"); }
   }

   public void go()
   {
      ...
      buttonCow.addActionListener(new CowListener());
      buttonDog.addActionListener(new DogListener());
      buttonCat.addActionListener(new CatListener());
   }
   ...
}
```

Panels

- JPanel
  - Widget that you can draw lines, circles, images, etc.
    - Like StdDraw
  - Needs to be added to a JFrame
  - A class that extends JPanel, drawing done by:
    - `public void paintComponent(Graphics g)`
    - Called automatically when needed (e.g. resize of window)
    - By calling repaint() on JFrame



```
public class Panel
{
   public static void main(String [] args)
   {
      JFrame frame = new JFrame();
      MyDrawPanel panel = new MyDrawPanel();

      frame.getContentPane().add(BorderLayout.CENTER, panel);
      frame.setSize(300, 300);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
   }
}
```

```
public class MyDrawPanel extends JPanel
{
   public void paintComponent(Graphics g)
   {
      g.setColor(Color.orange);
      g.fillRect(20,50,100,100);

      g.setColor(new Color(1.0f, 0.0f, 1.0f));
      g.drawLine(0, 0, 100, 100);

      Image image = new ImageIcon("cat.jpg").getImage();
      g.drawImage(image, 150, 150, this);
   }
}
```

Summary

- Building Java GUIs
  - `JFrame` basis for GUI instead of CLI interfaces
  - Choice of layout managers
    - We looked at just three:
      - `BorderLayout, FlowLayout, BoxLayout`
    - Handles where widgets such as buttons appear
  - Event handling
    - Widgets trigger events
    - Notify any registered listeners
      - Single listener that handles multiple widget
      - Separate inner class for each widget
  - Drawing primitive shapes, images
    - Create a class that `extends JPanel`