The University Of Michigan

EECS402

Intro To Java

Andrew M. Morgan

Andrew M Morgan
Reading: 1 (or more) of approximately 140000 Java Books

Suggested: Java Software Solutions
by John Lewis and William Loftus
(Addison Wesley)

Java Pros

- Pros Vs. C++
  - "Compiled" Java is platform independent
    - Java is compiled to "bytecodes" instead of object code
    - Executing machine has a "Java Virtual Machine" (JVM) that interprets and executes the bytecodes
  - Simpler (arguably)
  - No pointers (kind of)
  - Association with Web using Applets
  - Automatic garbage collection
  - Automatic array bounds checking
  - A gazillion classes available for use, described in nice API
    - http://java.sun.com/j2se/1.5.0/docs/api/
  - No globals - everything's a member - even main()!
  - Straight forward creation and use of graphical interfaces
  - Great for prototyping

Java Cons

- Cons Vs. C++
  - Java is compiled to Bytecodes
    - JVM is needed on executing machine
    - JVM interpreting byte codes is slow(er)
  - Automatic garbage collection (GC)
    - Extra runtime work slows things down
    - C++ programmer must manage memory without GC overhead
  - Automatic array bounds checking
    - Extra runtime work slows things down
  - A gazillion classes available for use, described in nice API
    - Finding what you are looking for is a challenge sometimes
    - Even if you find something, there might be something better
  - No globals - everything's a member - even main()!
    - Sometimes globals make sense. Even constants have to be members of a class.....

Hello World In Java


```
import java.io.*;

class HelloWorldClass
{
  public static void main(
       String args[]
       )
  {
    System.out.println("Hello World!");
  }
}
```

Similar to "#include <iostream>"

main function (it's a member)

command line args (like argv[])

Like "cout <<"

```
prompt% javac HelloWorldClass.java
prompt% java HelloWorldClass
Hello World!
prompt%
```

Compiles Java source code into bytecodes, output file is "HelloWorldClass.class"

Runs Java Virtual Machine to interpret and execute bytecodes in file HelloWorldClass.class

Execution Output

Java Has "No Pointers"

- Some people will say Java has no pointers
  - Instead, except for simple data types like int, float, etc, *everything* in Java is a "reference"
  - What's a "reference"? → A pointer to data
- In reality, everything is a pointer
  - In C++ you might have an actual object, or you might have a pointer to an object
  - Java only provides one option, so there is no confusion
- If you fail to instantiate a reference, using the "new" operator, then access the reference, you get a "NullPointerException"

Arrays

- Arrays in Java are not "simple types" and therefore are stored as references
- Declare a reference using [] notation
- Use new to allocate actual array
- Array reference has a "length" member to provide number of elements in array being referenced
- Java does auto-bounds checking
  - If you index out of the valid bounds of an array, an exception is thrown
    - ArrayIndexOutOfBoundsException
  - You can "catch" (and handle) these exceptions if you wish
  - If not caught, will cause program to end (like a seg fault)

Arrays, Example


```
import java.io.*;

class Array1Class
{
  public static void main(
        String args[]
        )
  {
    //int intAry[10];
    int intAry[];
    int i;

    intAry = new int[10];
    for (i = 0; i < intAry.length; i++)
    {
      intAry[i] = (i + 3) * i;
      System.out.println("Val: " + i + ": " + intAry[i]);
    }
    intAry[12] = 12;
    System.out.println("This is never printed");
  }
}
```

Not allowed in Java!!! When you declare an array, you declare a *reference* and must use "new" to allocate the array!

Correctly declares a reference to an array of integers

Allocates integer array, and sets "intAry" reference to refer to allocated array

Array lengths can be determined using length member of array reference.

"+" operator on strings can concatenate lots of data types to a string

Throws an ArrayIndexOutOfBoundsException

Statement is not reached due to exception being thrown, but not caught...

Arrays, Example Output


```
Val: 0: 0
Val: 1: 4
Val: 2: 10
Val: 3: 18
Val: 4: 28
Val: 5: 40
Val: 6: 54
Val: 7: 70
Val: 8: 88
Val: 9: 108
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 12
        at HelloWorldClass.main(HelloWorldClass.java:19)
```

Arrays, Example2


```
import java.io.*;
class Array2Class
{
  public static void main(
       String args[]
       )
  {
    int intAry[];
    int i;

    intAry = new int[10];
    for (i = 0; i < intAry.length; i++)
    {
      intAry[i] = (i + 3) * i;
      System.out.println("Val: " + i + ": " + intAry[i]);
    }
    try
    {
      intAry[12] = 12;
    }
    catch (ArrayIndexOutOfBoundsException e)
    {
      System.out.println("Caught OutOfBounds Excep!");
    }
    System.out.println("This is printed now!");
  }
}
```

Val: 0: 0
Val: 1: 4
Val: 2: 10
Val: 3: 18
Val: 4: 28
Val: 5: 40
Val: 6: 54
Val: 7: 70
Val: 8: 88
Val: 9: 108
Caught OutOfBounds Excep!
This is printed now!

Try says "the following might throw an exception, which I might be able to catch"

Throws an ArrayIndexOutOfBoundsException

Catch catches the appropriate exception type, and handles it in some way, allowing program to continue...

Statement is reached since the exception was caught

Java's Strict Compiler

- Java's compiler "helps you" by checking whether a variable has been set when its used

```
import java.io.*;
class StrictCompileClass
{
  public static void main(
       String args[]
       )
  {
    int i;
    int j;
    i = 4;
    if (i < 0)
    {
      j = -1;
    }
    else if (i > 0)
    {
      j = 1;
    }
    else if (i == 0)
    {
      j = 0;
    }
    System.out.println("J is: " + j );
  }
}
```

```
StrictCompileClass.java:24: variable j might not have been initialized
   System.out.println("J is: " + j );
                                  ^
1 error
```

Dealing With Java's Strict Compiler

- Both of the following are acceptable fixes for prior problem

```
public static void main(
        String args[]
        )
  {
    int i;
    int j;
    i = 4;
    j = 0;
    if (i < 0)
    {
      j = -1;
    }
    else if (i > 0)
    {
      j = 1;
    }
    else if (i == 0)
    {
      j = 0;
    }
    System.out.println("J is: " + j );
  }
```

```
public static void main(
        String args[]
        )
  {
    int i;
    int j;

    i = 4;
    if (i < 0)
    {
      j = -1;
    }
    else if (i > 0)
    {
      j = 1;
    }
    else
    {
      j = 0;
    }
    System.out.println("J is: " + j );
  }
```


Classes, Ctors, Members, Etc.


```
import java.lang.*;

public class CircleClass
{
  private double radius;

  public CircleClass(
        double inRad
        )
  {
    radius = inRad;
  }

  public double computeArea(
        )
  {
    return (Math.PI * radius * radius);
  }
}
```

Allows use of "Math"

Private attribute

ctor like C++

Member function

```
import java.io.*;

public class MyCirclesClass
{
  public static void main(
        String args[]
        )
  {
    CircleClass circ1;
    CircleClass circ2;
    double totalArea;

    circ1 = new CircleClass(3.5);
    circ2 = new CircleClass(4.0);

    totalArea = circ1.computeArea() +
                circ2.computeArea();
    System.out.println("Total Area: " +
                       totalArea);
  }
}
```

References only!

```
[ 68 ] java -: javac CircleClass.java
[ 69 ] java -: javac MyCirclesClass.java
[ 70 ] java -: java MyCirclesClass
Total Area: 88.74999246391165
[ 71 ] java -:
```

Writes CircleClass.class
Writes MyCirclesClass.class
Runs main() in MyCirclesClass.class
Program Output

Interfaces and Adapters

- Java has a concept referred to as an "interface"
  - Its just an abstract class – a class that has interfaces to functions, but no implementations
  - Just like C++, you create a class that provides implementations
    - In Java, this is called "implementing the interface"
    - Java sees this as being different from inheritance
- An "adapter" is a class that is derived from an interface that provides stub (empty) functions for all interface methods
  - Now, if you only need one function in the interface, you can derive from the adapter and override just the one function of interest
  - There is no need to provide stubs for all others, because the adapter has already done it
- You can implement as many interfaces as you want, but can only inherit from one
  - Java does *not* support multiple inheritance

Java GUI Support

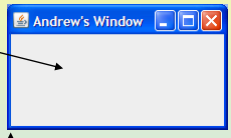
- Ability to easily create a Graphical User Interface is one reason Java is so popular
  - AWT: Abstract Windowing Toolkit
    - The early Java way to create GUIs
    - Very easy and (mostly) intuitive
    - AWT stuff is in java.awt.<stuff>
  - Swing
    - Newer GUI toolkit
    - Built on top of AWT, and lots of similar functionality to AWT
    - Swing stuff is in javax.swing.<stuff>
    - Swing components start with a J to distinguish from AWT counterparts
- This lecture will talk mostly about Swing – older existing programs will use AWT though

Creating a Frame

- A window is called a Frame in the AWT and a JFrame in Swing
- Instantiate a JFrame, and set it to visible, and voila – a Frame
  - By default, clicking "X" will hide the window, but not exit the program

```
package eecs285.test;

import java.awt.FlowLayout;
import javax.swing.JLabel;
import javax.swing.JFrame;
import javax.swing.JButton;

public class JFrameDemo
{
  public static void main(String [] args)
  {
    JFrame win;
    win = new JFrame("Andrew's Window");
    win.setSize(220, 125);
    win.setVisible(true);
    win.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
}
```

No window contents were specified, therefore the window is empty



Window decorations and border are automatic

Window title given to ctor

Displays the frame on the screen

Causes application to exit when "X" is clicked

JLabel, JButton, and FlowLayout

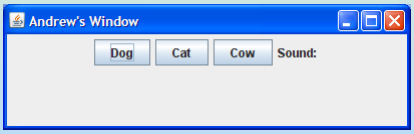
- A JLabel is just a bit of text that is displayed
  - User doesn't modify a label directly
- JButton is a button that can be clicked on
- When designing an interface, decide how you want the components displayed
  - Several different "layout managers" available for common ways
  - FlowLayout adds a component immediately to the right of the previous component, until there's no more room, then the component is placed on the next row
    - Works like a typewriter, but for Swing components
- These components (and other components) can be added to a layout (in a frame, or JPanel, etc) with the add method

Using Buttons and Labels


```
import java.awt.FlowLayout;
import javax.swing.JLabel;
import javax.swing.JFrame;
import javax.swing.JButton;
public class JFrameDemo
{
  public static void main(String [] args)
  {
    JFrame win;
    JButton dogButton;
    JButton catButton;
    JButton cowButton;
    JLabel label;

    win = new JFrame("Andrew's Window");
    win.setSize(220, 125);
    win.setLayout(new FlowLayout());
    dogButton = new JButton("Dog");
    win.add(dogButton);
    catButton = new JButton("Cat");
    win.add(catButton);
    cowButton = new JButton("Cow");
    win.add(cowButton);
    label = new JLabel("Sound: ");
    win.add(label);
    win.setVisible(true);
    win.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
}
```

Nice buttons – but they don't do anything when you click them…..

FlowLayout will fit as many components on one row before starting another, so if you resize the above window, it looks like this:



or this:

Event Handling

- Java uses an event handling scheme for dealing with users interacting with the components
- When a user clicks a button (for example), an event is triggered
- If you don't listen for the event, and write some code to do something with it, your button is still useless
- There are several different types of "event listeners"
  - When a user clicks a mouse button, a MouseEvent is triggered, and you can handle it by implementing a MouseListener interface
  - There are also keyboard listeners, window listeners, etc
- After implementing the MouseListener interface, you have to register the listener with a component, so it knows it should be listening for events
- Since listeners are very tightly bound to the interface they apply to, they are usually implemented as inner classes

Customize JFrame

- Before we implement the event listener, let's reorganize
  - Make a new class that inherits from JFrame
  - Put the UI components as member data to the custom frame
  - Put the UI setup in the ctor
- Main is fairly straight forward

```
package eecs285.test;

import java.awt.FlowLayout;
import javax.swing.JFrame;

public class AndrewsFrameDemo
{
  public static void main(String [] args)
  {
    AndrewsFrame win;

    win = new AndrewsFrame("Andrew's Window");
    win.setSize(220, 125);
    win.setVisible(true);
    win.setDefaultCloseOperation(
                    JFrame.EXIT_ON_CLOSE);
  }
}
```

Recall use of super to pass parameters on to the parent class (JFrame here) ctor

```
package eecs285.test;

import java.awt.FlowLayout;
import javax.swing.JLabel;
import javax.swing.JFrame;
import javax.swing.JButton;
import static java.lang.System.out;

public class AndrewsFrame extends JFrame
{
  private JButton dogButton;
  private JButton catButton;
  private JButton cowButton;
  private JLabel label;

  public AndrewsFrame(String inTitle)
  {
    super(inTitle);
    setLayout(new FlowLayout());
    dogButton = new JButton("Dog");
    add(dogButton);
    catButton = new JButton("Cat");
    add(catButton);
    cowButton = new JButton("Cow");
    add(cowButton);
    label = new JLabel("Sound: ");
    add(label);
  }
}
```


Adding Mouse Listener

- MouseListener is an interface with 5 methods that need to be implemented
  - mouseClicked: Triggered when mouse button is pressed and released on the same component, without moving off the component in between
  - mouseEntered: Triggered when the mouse cursor enters the bounds of the component
  - mouseExited: Triggered when the mouse cursor exits the bounds of the component
  - mousePressed: Triggered when the mouse button is pressed within the bounds of a component
  - mouseReleased: Triggered when the mouse button is released within the bounds of a component
- Inside the AndrewsFrame definition, create an inner class that implements the MouseListener interface

Initial MouseListener Added

- For now, triggered mouse events just print a message to console

```
public class AndrewsFrame extends JFrame
{
  private JButton dogButton;
  private JButton catButton;
  private JButton cowButton;
  private JLabel label;
  private AndrewsListener mouseListener;

  public AndrewsFrame(String inTitle)
  {
    super(inTitle);
    setLayout(new FlowLayout());
    dogButton = new JButton("Dog");
    add(dogButton);
    catButton = new JButton("Cat");
    add(catButton);
    cowButton = new JButton("Cow");
    add(cowButton);
    label = new JLabel("Sound: ");
    add(label);

    mouseListener = new AndrewsListener();
    dogButton.addMouseListener(mouseListener);
    catButton.addMouseListener(mouseListener);
    cowButton.addMouseListener(mouseListener);
  }

  public class AndrewsListener implements MouseListener
  {
```

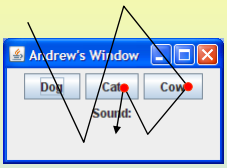
//cot'd next column

```
    public void mouseClicked(MouseEvent e)
    {
      out.println("Component clicked");
    }
    public void mouseEntered(MouseEvent e)
    {
      out.println("Component Entered");
    }
    public void mouseExited(MouseEvent e)
    {
      out.println("Component Exited");
    }
    public void mousePressed(MouseEvent e)
    {
      out.println("Component Pressed");
    }
    public void mouseReleased(MouseEvent e)
    {
      out.println("Component Released");
    }
  } //end class AndrewsListener
} //end class AndrewsFrame
```

Registers the listener to each of the button components. Any MouseEvent that occurs on any of the buttons will cause the appropriate listener method to be called

Events Being "Handled"

- Buttons still cause no change to interface
- Output is sent to console now, though



Line represents the path of the mouse cursor, and red dots represent click locations

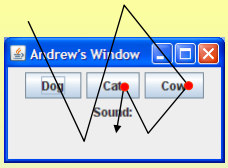
Component Entered	dog
Component Exited	
Component Entered	cat
Component Exited	
Component Entered	cow
Component Pressed	
Component Released	
Component clicked	
Component Exited	
Component Entered	cat
Component Pressed	
Component Released	
Component clicked	
Component Exited	

Using MouseAdapter

- Quite often, the events you want to deal with are limited
- When you implement the MouseListener interface, all 5 methods need to be implemented
  - Even if you don't want to do anything, you must make an empty method
- The MouseAdapter class is a simple class that contains empty methods for all 5 methods in the MouseListener interface
  - By extending the MouseAdapter class, you simply override the function(s) you want to, and leave the others empty as specified by the parent (MouseAdapter)
  - Much easier than remembering every method name, or copy/pasting from the API every time

"implements MouseListener" to "extends MouseAdapter"


```
public class AndrewsFrame extends JFrame
{
  private JButton dogButton;
  private JButton catButton;
  private JButton cowButton;
  private JLabel label;
  private AndrewsListener mouseListener;

  public AndrewsFrame(String inTitle)
  {
    super(inTitle);
    setLayout(new FlowLayout());
    dogButton = new JButton("Dog");
    add(dogButton);
    catButton = new JButton("Cat");
    add(catButton);
    cowButton = new JButton("Cow");
    add(cowButton);
    label = new JLabel("Sound: ");
    add(label);

    mouseListener = new AndrewsListener();
    dogButton.addMouseListener(mouseListener);
    catButton.addMouseListener(mouseListener);
    cowButton.addMouseListener(mouseListener);
  }
  public class AndrewsListener extends MouseAdapter
  {
    public void mouseClicked(MouseEvent e)
    {
      out.println("Component clicked");
    }
```



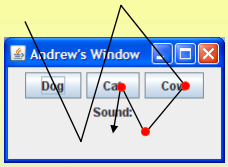
Component clicked
Component clicked

Using the MouseAdapter allows you to only implement the methods for event types you want to handle.

mouseEntered, mouseReleased, etc., are now essentially ignored

Best Choice: Using ActionListener

- MouseEvents can be triggered on any Swing component, including the JFrame
- Can be confusing with lots of events being triggered
- ActionEvents are triggered when a JButton is pressed
  - Next slide shows use of ActionListener




```
    ... same definition, members, and other statements before
    mouseListener = new AndrewsListener();
    dogButton.addMouseListener(mouseListener);
    catButton.addMouseListener(mouseListener);
    cowButton.addMouseListener(mouseListener);
    this.addMouseListener(mouseListener);
  }

  public class AndrewsListener extends MouseAdapter
  {
    public void mouseClicked(MouseEvent e)
    {
      out.println("Button clicked");
    }
    public void mouseEntered(MouseEvent e)
    {
      out.println("Button entered");
```

← Registers the listener to the frame as well

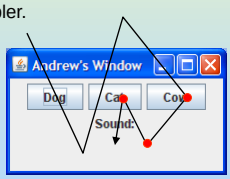
Component entered
Component entered
Component entered
Component entered
Component entered
Component entered
Component entered
Component clicked
Component entered
Component entered
Component clicked
Component entered

Switching to ActionListener


```
//... same definition and data members as before
public AndrewsFrame(String inTitle)
{
  super(inTitle);
  setLayout(new FlowLayout());
  dogButton = new JButton("Dog");
  add(dogButton);
  catButton = new JButton("Cat");
  add(catButton);
  cowButton = new JButton("Cow");
  add(cowButton);
  label = new JLabel("Sound: ");
  add(label);

  actionListener = new AndrewsListener();
  dogButton.addActionListener(actionListener);
  catButton.addActionListener(actionListener);
  cowButton.addActionListener(actionListener);
  //this.addActionListener(actionListener);
}

public class AndrewsListener implements ActionListener
{
  public void actionPerformed(ActionEvent e)
  {
    out.println("Action was performed");
  }
}
```

```
public class AndrewsFrameDemo
{
  public static void main(String [] args)
  {
    AndrewsFrame win;

    win = new AndrewsFrame("Andrew's Window");
    win.setSize(220, 125);
    win.setVisible(true);
    win.setDefaultCloseOperation(
                    JFrame.EXIT_ON_CLOSE);
  }
}
```

ActionEvents are not triggered due to interaction with a JFrame, so this functionality is not supported.
Choosing the right event types makes everything simpler.



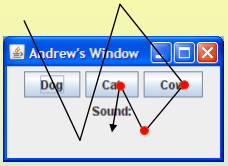
Action was performed
Action was performed

ActionListener has only one method – actionPerformed – therefore no "ActionAdapter" is necessary or available

Which Button?

- How to determine which button was pressed?
  - Every event has a getSource() method
    - Returns a reference to the object that triggered the event

```
... Rest of "AndrewsFrame" is unchanged
public class AndrewsListener implements ActionListener
{
  public void actionPerformed(ActionEvent e)
  {
    if (e.getSource() == dogButton)
    {
      out.println("Dog Button Pressed");
    }
    else if (e.getSource() == catButton)
    {
      out.println("Cat Button Pressed");
    }
    else if (e.getSource() == cowButton)
    {
      out.println("Cow Button Pressed");
    }
  }
} //end class AndrewsListener
//end class AndrewsFrame
```

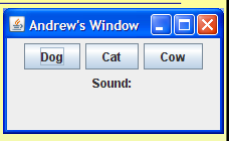


Cow Button Pressed
Cat Button Pressed

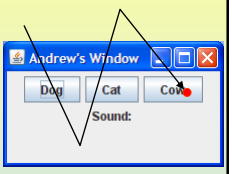
Updating Label on Button Press

- Often Java console is not displayed
- Update a component on the UI instead
  - JLabel has a "setText" method

```
... Rest of "AndrewsFrame" is unchanged
public class AndrewsListener implements ActionListener
{
  public void actionPerformed(ActionEvent e)
  {
    if (e.getSource() == dogButton)
    {
      label.setText("Sound: Woof");
    }
    else if (e.getSource() == catButton)
    {
      label.setText("Sound: Meow");
    }
    else if (e.getSource() == cowButton)
    {
      label.setText("Sound: Mooo");
    }
  }
} //end class AndrewsListener
//end class AndrewsFrame
```



User moves mouse, clicks on Cow
On click,



label is updated.
User moves mouse, clicks on Cat.
On click,

label is updated.

Adding Icons to JButtons

- Any button can have text and/or an icon
  - Just pass the name of an image to the JButton ctor at instantiation

```
... everything else the same
public AndrewsFrame(String inTitle)
{
  super(inTitle);
  setLayout(new FlowLayout());
  dogButton = new JButton(new ImageIcon("dogIcon.jpg"));
  add(dogButton);
  catButton = new JButton(new ImageIcon("catIcon.jpg"));
  add(catButton);
  cowButton = new JButton(new ImageIcon("cowIcon.jpg"));
  add(cowButton);
  label = new JLabel("Sound: ");
  add(label);
... everything else the same
```

Click on the dog button to get:

The Whole Program

Here's the fully modified program to this point

```
package eecs285.test;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.FlowLayout;
import javax.swing.JLabel;
import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.ImageIcon;
import static java.lang.System.out;

public class AndrewsFrame extends JFrame
{
  private JButton dogButton;
  private JButton catButton;
  private JButton cowButton;
  private JLabel label;
  private AndrewsListener actionListener;

  public AndrewsFrame(String inTitle)
  {
    super(inTitle);
    setLayout(new FlowLayout());
    dogButton = new JButton(new ImageIcon("dogIcon.jpg"));
    add(dogButton);
    catButton = new JButton(new ImageIcon("catIcon.jpg"));
    add(catButton);
    cowButton = new JButton(new ImageIcon("cowIcon.jpg"));
    add(cowButton);
    label = new JLabel("Sound: ");
    add(label);

    actionListener = new AndrewsListener();
    dogButton.addActionListener(actionListener);
    catButton.addActionListener(actionListener);
    cowButton.addActionListener(actionListener);
```

```
public class AndrewsListener implements ActionListener
{
  public void actionPerformed(ActionEvent e)
  {
    if (e.getSource() == dogButton)
    {
      label.setText("Sound: Woof");
    }
    else if (e.getSource() == catButton)
    {
      label.setText("Sound: Meow");
    }
    else if (e.getSource() == cowButton)
    {
      label.setText("Sound: Mooo");
    }
  }
} //end class AndrewsListener
} //end class AndrewsFrame
```

```
package eecs285.test;

import java.awt.FlowLayout;
import javax.swing.JFrame;

public class AndrewsFrameDemo
{
  public static void main(String [] args)
  {
    AndrewsFrame win;

    win = new AndrewsFrame("Andrew's Window");
    win.setSize(220, 125);
    win.setVisible(true);
    win.setDefaultCloseOperation(
                JFrame.EXIT_ON_CLOSE);
  }
}
```

Making GUIs Look Like They're Supposed To

- What if I have a certain layout of my GUI components I need to implement?
- For example, if I was given this as a spec and was required to make the GUI look exactly like this, how to do it?

This is the goal GUI look.

Obviously, the initial GUI was not laid out this way.

Other Layout Managers

- FlowLayout is the only layout manager discussed so far
  - Added components flow left-to-right, until there's no more room, then they start a new row and continue left-to-right
  - Things move around when the window is resized, etc.
- GridLayout is useful at times
  - Specify exactly how many rows and columns of components you want
  - Each grid space is the same size, so some components can come out looking out of proportion
- BorderLayout is, in my opinion, most commonly the most useful
  - Components are "anchored" to the top (north), right (east), bottom (south), left (west), or middle (center) of the area
- BoxLayout
  - Aligns components horizontally or vertically

Using Layouts

- It is nearly impossible to use a single layout manager and get the GUI looking the way you want it
- JPanel is a swing component that can be placed anywhere any other component can be placed
- JPanel can have their own layout managers and can contain components
- Use JPanels and multiple layout managers to position the components the way you want them
  - Takes some significant practice to get it all straight

BorderLayout By Itself

- Here is what results from trying to use BorderLayout by itself:

```
public AndrewsFrameBorder(String inTitle)
{
  super(inTitle);
  setLayout(new BorderLayout());
  dogButton = new JButton(new ImageIcon("dogIcon.jpg"));
  add(dogButton, BorderLayout.NORTH);
  catButton = new JButton(new ImageIcon("catIcon.jpg"));
  add(catButton, BorderLayout.EAST);
  cowButton = new JButton(new ImageIcon("cowIcon.jpg"));
  add(cowButton, BorderLayout.WEST);
  label = new JLabel("Sound: ");
  add(label, BorderLayout.SOUTH);

  actionListener = new AndrewsListener();
  dogButton.addActionListener(actionListener);
  catButton.addActionListener(actionListener);
  cowButton.addActionListener(actionListener);
}
```

Dog button takes up the entire "north" region, Sound label is small and off to the left.

Note: JLabel can be told to center the text, which then looks like this:

GridLayout By Itself

- GridLayout is good, but it forces each grid space to be exactly the same size.
  - The "Sound: " label is forced to fit in the same space as a button

```
public AndrewsFrameGrid(String inTitle)
{
  super(inTitle);
  setLayout(new GridLayout(3, 3));
  dogButton = new JButton(new ImageIcon("dogIcon.jpg"));
  add(new JLabel(""));
  add(dogButton);
  add(new JLabel(""));
  catButton = new JButton(new ImageIcon("catIcon.jpg"));
  add(catButton);
  add(new JLabel(""));
  cowButton = new JButton(new ImageIcon("cowIcon.jpg"));
  add(cowButton);
  label = new JLabel("Sound: ", SwingConstants.CENTER);
  add(new JLabel(""));
  add(label);
  add(new JLabel(""));

  actionListener = new AndrewsListener();
  dogButton.addActionListener(actionListener);
  catButton.addActionListener(actionListener);
  cowButton.addActionListener(actionListener);
```

Label is cut off because it is forced into a grid space the same size as the icons. Cat and cow are spread too far apart.

Using GridLayout *and* FlowLayout


```
public AndrewsFrameFinal(String inTitle)
{
  super(inTitle);

  JPanel topPan = new JPanel(new FlowLayout());
  JPanel middlePan = new JPanel(new FlowLayout());
  JPanel bottomPan = new JPanel(new FlowLayout());

  setLayout(new GridLayout(3, 1));

  dogButton = new JButton(new ImageIcon("dogIcon.jpg"));
  topPan.add(dogButton);

  catButton = new JButton(new ImageIcon("catIcon.jpg"));
  middlePan.add(catButton);
  cowButton = new JButton(new ImageIcon("cowIcon.jpg"));
  middlePan.add(cowButton);

  label = new JLabel("Sound: ", SwingConstants.CENTER);
  bottomPan.add(label);

  add(topPan);
  add(middlePan);
  add(bottomPan);

  actionListener = new AndrewsListener();
  dogButton.addActionListener(actionListener);
  catButton.addActionListener(actionListener);
  cowButton.addActionListener(actionListener);
}
```

So Close – but the bottoms of the buttons seems to be cut off……..

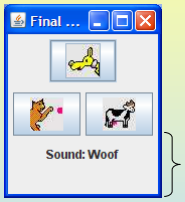
Problem is setSize() – the specified height is not sufficient for the window!

```
public class AndrewsFrameFinalDemo
{
  public static void main(String [] args)
  {
    AndrewsFrameFinal win;

    win = new AndrewsFrameFinal("Final Window");
    win.setSize(220, 125);
    win.setVisible(true);
    win.setDefaultCloseOperation(
                JFrame.EXIT_ON_CLOSE);
  }
}
```

Using pack() Instead of setSize()

- Method pack will fit all the components that have been added and make the frame the size needed

```
public class AndrewsFrameFinalDemo
{
  public static void main(String [] args)
  {
    AndrewsFrameFinal win;

    win = new AndrewsFrameFinal("Final Window");
    //win.setSize(220, 125);
    win.pack();
    win.setVisible(true);
    win.setDefaultCloseOperation(
                          JFrame.EXIT_ON_CLOSE);
  }
}
```



The bottom looks kind of .. weird. The reason is with Grid layout, each grid is the same size, so the 3rd row is the same height as the other rows, which isn't necessary to store the label.

BorderLayout, GridLayout, *and* FlowLayout


```
public AndrewsFrameFinal(String inTitle)
{
  super(inTitle);

  JPanel centerPan = new JPanel(new GridLayout(2, 1));
  JPanel southPan = new JPanel(new FlowLayout());
  JPanel centerTopPan = new JPanel(new FlowLayout());
  JPanel centerBotPan = new JPanel(new FlowLayout());

  setLayout(new BorderLayout());

  dogButton = new JButton(new ImageIcon("dogIcon.jpg"));
  centerTopPan.add(dogButton);

  catButton = new JButton(new ImageIcon("catIcon.jpg"));
  centerBotPan.add(catButton);
  cowButton = new JButton(new ImageIcon("cowIcon.jpg"));
  centerBotPan.add(cowButton);

  centerPan.add(centerTopPan);
  centerPan.add(centerBotPan);

  label = new JLabel("Sound: ", SwingConstants.CENTER);
  southPan.add(label);

  add(centerPan, BorderLayout.CENTER);
  add(southPan, BorderLayout.SOUTH);

  actionListener = new AndrewsListener();
  dogButton.addActionListener(actionListener);
  catButton.addActionListener(actionListener);
  cowButton.addActionListener(actionListener);
```

Finally – we've got what we need – like I said: It takes some serious practice…

Final Layout Description

Red Box: CENTER panel of Frame's BorderLayout
Green Box: SOUTH panel of Frame's BorderLayout
Note: NORTH, EAST, and WEST unused

Cyan Line: Separator between rows of the Red Box's GridLayout

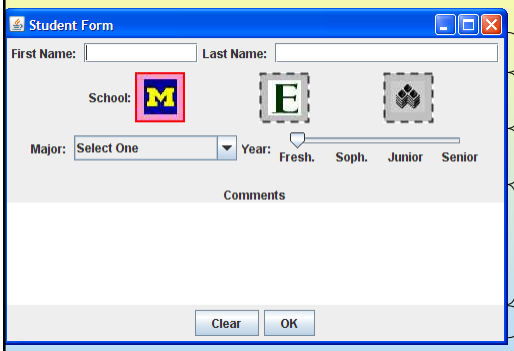
Note: As I resize the window, GUI look is maintained

Other UI JComponents

- Once you understand layout managers, and how JComponents work, adding other UI components is easy
  - JButton: Click-able button with icon and/or text
  - JCheckBox: Toggle-able box with label
  - JComboBox: Drop-down list of items user can choose
  - JFileChooser: Standard interface for selecting a file from the file system
  - JList: Listed items user can choose
    - similar to combo box, but not drop-down
  - JMenuBar: Drop-down menus (File, Help, etc)
  - JPopupMenu: Pop-up for context-dependent menus on a component
  - JRadioButton: Group of select-able buttons, only one of which can be selected at a time
  - JScrollPane: Allows a component to fit into a smaller space with scroll bars around it
  - JSlider: Slider bar to specify a value within a range
  - JTabbedPane: Supports tabbed interface for switch-able pages within a single frame
  - JTextField: Allows user to type small amount of text to UI
  - JTextArea, JEditorPane, JTextPane: Allows larger amounts of text to be entered
  - JToolBar: Move-able and attach-able selection of controls on a bar
  - JToolTip: Support for pop-up info on mouse hover-over a component
  - Others too – see the API

Example With Other Components

namePanel, in the NORTH part of the JFrame's BorderLayout Consists of two JLabels and two JTextFields



schoolPanel, in the first row of the centerTopPanel's GridLayout. Consists of a JLabel, and one ButtonGroup with 3 icon-style JRadioButtons

centerTopPanel, in the first row of centerPanel's GridLayout. Consists of 2 sub-panels

centerPanel, in the CENTER part of the JFrame's BorderLayout. Consists of 2 sub-panels

majYearPanel, in the second row of the centerTopPanel's GridLayout. Consists of two JLabels, a JComboBox, and a JSlider

commentsPanel in the second row of centerPanel's GridLayout. Consists of a JLabel and JTextArea

buttonPanel, in the SOUTH part of the JFrame's BorderLayout. Consists of two JButtons

StudentFormFrame


```
package eecs285.test;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.Dictionary;
import java.util.Hashtable;
import static java.lang.System.out;

public class StudentFormFrame extends JFrame
{
  private JTextField firstNameField;
  private JTextField lastNameField;
  private JComboBox majorCombo;
  private ButtonGroup schoolGroup;
  private JRadioButton umSchoolRadio;
  private JRadioButton eastSchoolRadio;
  private JRadioButton wccSchoolRadio;
  Dictionary< Integer, JComponent > yearDict; //for slider labels
  private JSlider yearSlider;
  private JTextArea commentsArea;
  private JButton clearButton;
  private JButton okButton;
  private FormButtonListener buttonListener;
```

StudentFormFrame ctor


```
public StudentFormFrame(String inTitle)
{
  super(inTitle); //Always has to be first

  JPanel namePanel;
  JPanel schoolPanel;
  JPanel majYearPanel;
  JPanel commentPanel;
  JPanel buttonPanel;
  JPanel centerTopPanel;
  JPanel centerPanel;

  firstNameField = new JTextField(10);
  lastNameField = new JTextField(20);
  namePanel = new JPanel(new FlowLayout());
  namePanel.add(new JLabel("First Name: ", SwingConstants.RIGHT));
  namePanel.add(firstNameField);
  namePanel.add(new JLabel("Last Name: ", SwingConstants.RIGHT));
  namePanel.add(lastNameField);
```

Declaring sub-panels, and setting up the namePanel

cot'd

StudentFormFrame ctor, cot'd

Setting up the icon-style radio buttons and the schoolPanel

```
  umSchoolRadio = new JRadioButton(new ImageIcon("umLogoUnsel.jpg"));
  umSchoolRadio.setSelectedIcon(new ImageIcon("umLogoSel.jpg"));
  umSchoolRadio.setSelected(true);
  eastSchoolRadio = new JRadioButton(new ImageIcon("eastLogoUnsel.jpg"));
  eastSchoolRadio.setSelectedIcon(new ImageIcon("eastLogoSel.jpg"));
  wccSchoolRadio = new JRadioButton(new ImageIcon("wccLogoUnsel.jpg"));
  wccSchoolRadio.setSelectedIcon(new ImageIcon("wccLogoSel.jpg"));
  schoolGroup = new ButtonGroup();
  schoolGroup.add(umSchoolRadio);
  schoolGroup.add(eastSchoolRadio);
  schoolGroup.add(wccSchoolRadio);
  schoolPanel = new JPanel(new GridLayout(1, 4));
  schoolPanel.add(new JLabel("School:", SwingConstants.RIGHT));
  schoolPanel.add(umSchoolRadio);
  schoolPanel.add(eastSchoolRadio);
  schoolPanel.add(wccSchoolRadio);
```

cot'd

StudentFormFrame ctor, cot'd


```
majorCombo = new JComboBox();
majorCombo.addItem("Select One");
majorCombo.addItem("Computer Science");
majorCombo.addItem("Computer Engineering");
majorCombo.addItem("Mathematics");
majorCombo.addItem("Chemistry");
majorCombo.addItem("Mechanical Engineering");
majorCombo.addItem("Other");

yearSlider = new JSlider(0, 3);
yearDict = new Hashtable< Integer, JComponent >();
yearDict.put(0, new JLabel("Fresh."));
yearDict.put(1, new JLabel("Soph."));
yearDict.put(2, new JLabel("Junior"));
yearDict.put(3, new JLabel("Senior"));
yearSlider.setLabelTable(yearDict);
yearSlider.setSnapToTicks(true);
yearSlider.setPaintLabels(true);
yearSlider.setValue(0);

majYearPanel = new JPanel(new FlowLayout());
majYearPanel.add(new JLabel("Major: ", SwingConstants.RIGHT));
majYearPanel.add(majorCombo);
majYearPanel.add(new JLabel("Year: ", SwingConstants.RIGHT));
majYearPanel.add(yearSlider);
```

Setting up the major field of studey combo box, the year slider, and the majYearPanel

StudentFormFrame ctor, cot'd


```
commentsArea = new JTextArea(5, 25);
commentPanel = new JPanel(new BorderLayout());
commentPanel.add(new JLabel("Comments", SwingConstants.CENTER),
                 BorderLayout.NORTH);
commentPanel.add(commentsArea, BorderLayout.CENTER);

clearButton = new JButton("Clear");
okButton = new JButton("OK");
buttonPanel = new JPanel(new FlowLayout());
buttonPanel.add(clearButton);
buttonPanel.add(okButton);

setLayout(new BorderLayout());
add(namePanel, BorderLayout.NORTH);
centerTopPanel = new JPanel(new GridLayout(2, 1));
centerTopPanel.add(schoolPanel);
centerTopPanel.add(majYearPanel);
centerPanel = new JPanel(new GridLayout(2, 1));
centerPanel.add(centerTopPanel);
centerPanel.add(commentPanel);
add(centerPanel, BorderLayout.CENTER);
add(buttonPanel, BorderLayout.SOUTH);

buttonListener = new FormButtonListener();
clearButton.addActionListener(buttonListener);
okButton.addActionListener(buttonListener);
}//end of constructor
```

Setting up the comment text area, the commentPanel, and the buttonPanel.

Setting up the JFrame with all the panels and sub-panels.

Adding an action listener to the buttons to handle the clear or ok button clicks.

FormButtonListener Implementation


```
public class FormButtonListener implements ActionListener
{
  public void actionPerformed(ActionEvent e)
  {
    String schoolStr;
    JLabel yearLabel;

    if (e.getSource() == okButton)
    {
      if (umSchoolRadio.isSelected())
      {
        schoolStr = "Michigan";
      }
      else if (eastSchoolRadio.isSelected())
      {
        schoolStr = "Eastern";
      }
      else //if (wccSchoolRadio.isSelected())
      {
        schoolStr = "WCC";
      }
      yearLabel = (JLabel)(yearSlider.getLabelTable().get(
                                                yearSlider.getValue()));
      out.printf("%s %s is a %s at %s, majoring in %s\n",
                 firstNameField.getText(),
                 lastNameField.getText(),
                 yearLabel.getText(),
                 schoolStr,
                 majorCombo.getSelectedItem());
```

Performing what needs to be done when the OK button is clicked

```
} //end handling OK button click
```

cot'd

FormButtonListener Implementation, cot'd


```
    else if (e.getSource() == clearButton)
    {
      firstNameField.setText("");
      lastNameField.setText("");
      umSchoolRadio.setSelected(true);
      eastSchoolRadio.setSelected(false);
      wccSchoolRadio.setSelected(false);
      majorCombo.setSelectedIndex(0);
      yearSlider.setValue(0);
      commentsArea.setText("");
    } //end handling Clear button click
  } //end method actionPerformed
} //end class FormButtonListener
} //end class StudentFormFrame
```

Performing what needs to be done when the Clear button is clicked

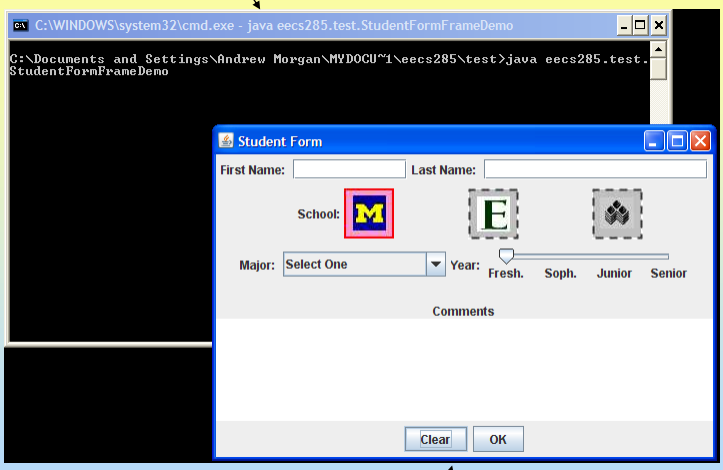
```
package eecs285.test;

public class StudentFormFrameDemo
{
  public static void main(String [] args)
  {
    StudentFormFrame win;

    win = new StudentFormFrame("Student Form");
    win.pack();
    win.setVisible(true);
    win.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Simple main program to display the GUI to the user

Using the StudentFormFrame

Console

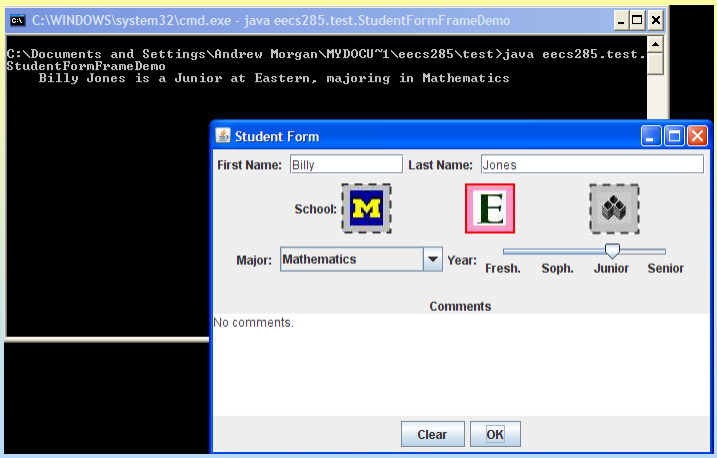
Initial State



StudentFormFrame

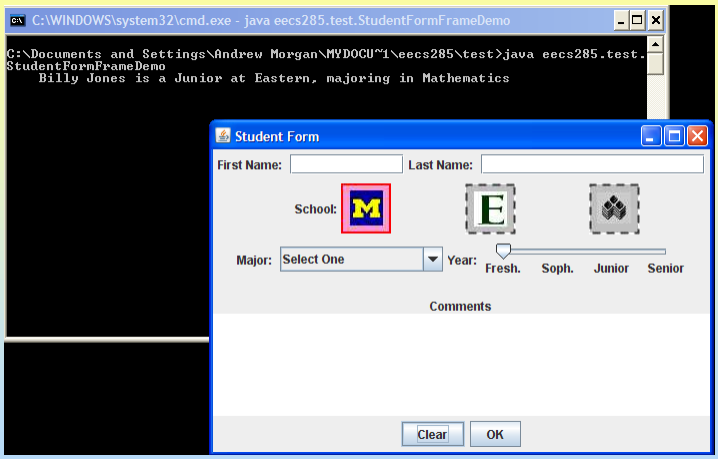
Using the StudentFormFrame, cot'd

Type Billy's info and click OK



Using the StudentFormFrame, cot'd

click Clear



Using the StudentFormFrame, cot'd

Type Julie's info and click OK