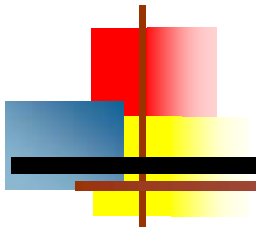
Networks and Internet Programming (0907522)

CHAPTER 2

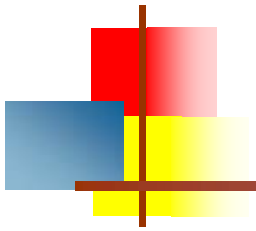
Java Overview

Instructor: Dr. Khalid A. Darabkh



Objectives

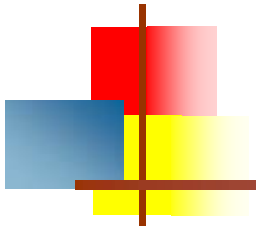
- The objectives of this chapter are:
  - To discuss the classes present in the java.awt package
  - To understand the inheritance hierarchy of the AWT
  - To outline the basic structure of GUIs
  - To show how to add components to containers
  - To understand how to use Layout Managers
  - To understand how to use Event Handling
  - To understand basic graphics processing under the AWT



AWT (Abstract Windowing Toolkit)

- The AWT is roughly broken into four categories
  - Components and Containers
  - Layout Managers
  - Event Handling
  - Graphics

- Many AWT components have been replaced by Swing components
- It is generally not considered a good idea to mix Swing components and AWT components. Choose to use one or the other.



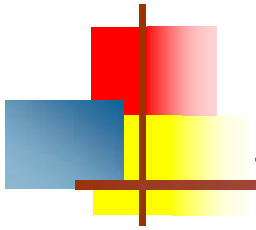
Running Java files

- **There are 3 different ways to run a java executable:**

- As a stand-alone program that can be invoked from the command line. This is termed an application.

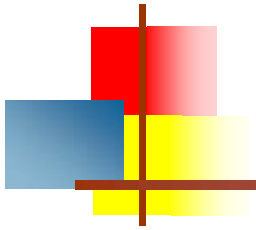
- As a program embedded in a web page to be run when the page is browsed. This is termed an applet.

- As a program that is invoked on demand on a server system and that runs in the context of a web server. This is termed a servlet.



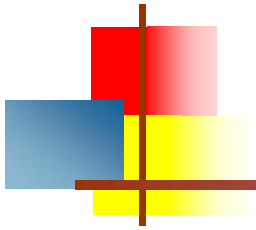
AWT Revisited

- Present in all Java implementations
- Described in most Java textbooks
- Adequate for many applications
- Uses the controls defined by your OS
  - therefore it's “least common denominator”
- Difficult to build an attractive GUI
- ```
  import java.awt.*;
  import java.awt.event.*;
  ```
  ```



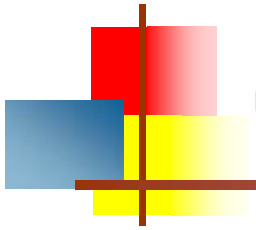
Swing

- Same concepts as AWT
- Doesn’t work in ancient Java implementations (Java 1.1 and earlier)
- Many more controls, and they are more flexible
  - Some controls, but not all, are a lot more complicated
- Gives a choice of “look and feel” packages
- Much easier to build an attractive GUI
- `import javax.swing.*;`



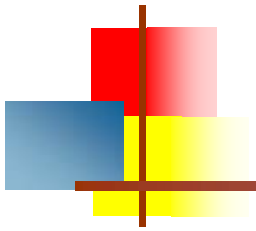
Swing vs. AWT

- Swing is bigger, slower, and more complicated
  - But not as slow as it used to be
- Swing is more flexible and better looking
- Swing and AWT are *incompatible*--you can use either, but you can't mix them
  - Actually, you can, but it's tricky and not worth doing
- Learning the AWT is a good start on learning Swing
- Many of the most common controls are just renamed
  - AWT: `Button b = new Button ("OK");`
    Swing: `JButton b = new JButton("OK");`



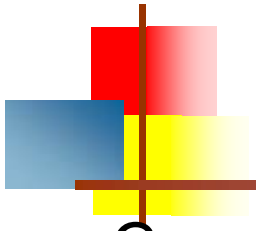
To build a GUI...

- Make somewhere to display things—usually a Frame or Dialog (for an application), or an Applet
- Create some Components, such as buttons, text areas, panels, etc.
- Add your Components to your display area
- Arrange, or *lay out*, your Components
- Attach Listeners to your Components
  - Interacting with a Component causes an Event to occur
  - A Listener gets a message when an interesting event occurs, and executes some code to deal with it



Containers and Components

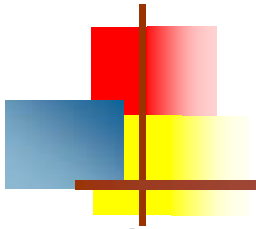
- The job of a Container is to hold and display Components
- Some common subclasses of Component are Button, Checkbox, Label, Scrollbar, TextField, **and** TextArea
- A Container is also a Component
  - This allows Containers to be nested
- Some Container subclasses are Panel (and Applet), Window, and Frame



Component

- Component is the superclass of most of the displayable classes defined within the AWT. Note: it is abstract.
- MenuComponent is another class which is similar to Component except it is the superclass for all GUI items which can be displayed within a drop-down menu.
- The Component class defines data and methods which are relevant to all Components

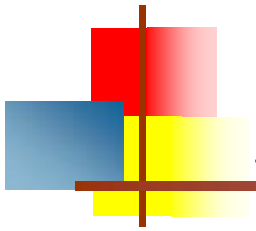
paint(Graphics g)
setBounds
setSize
setLocation
setFont
setEnabled
setVisible
setForeground **-- colour**
setBackground **-- colour**



Container

- Container is a subclass of Component. (ie. All containers are themselves, Components)
- Containers contain components
- For a component to be placed on the screen, it must be placed within a Container
- The Container class defined all the data and methods necessary for managing groups of Components

  add
  remove
  Validate
  invalidate
  setLayout

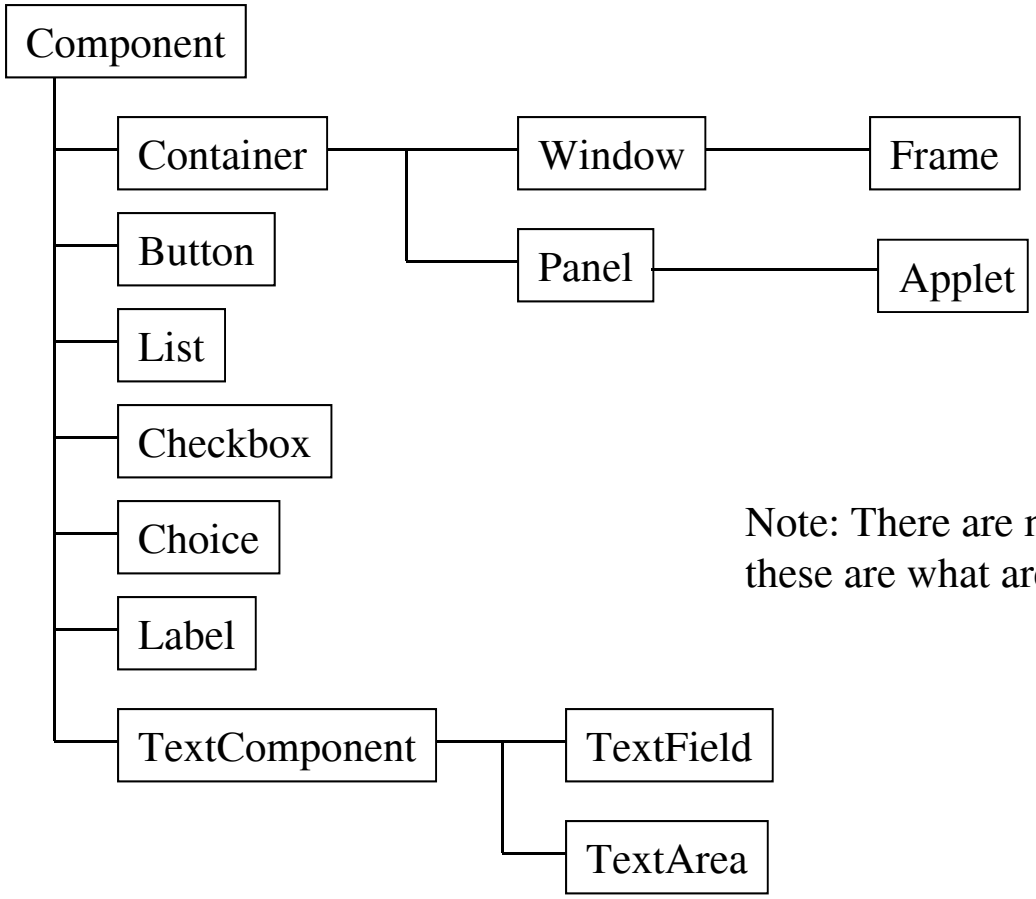


An Applet is Panel is a Container

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Panel
                           |
                           +----java.applet.Applet
```

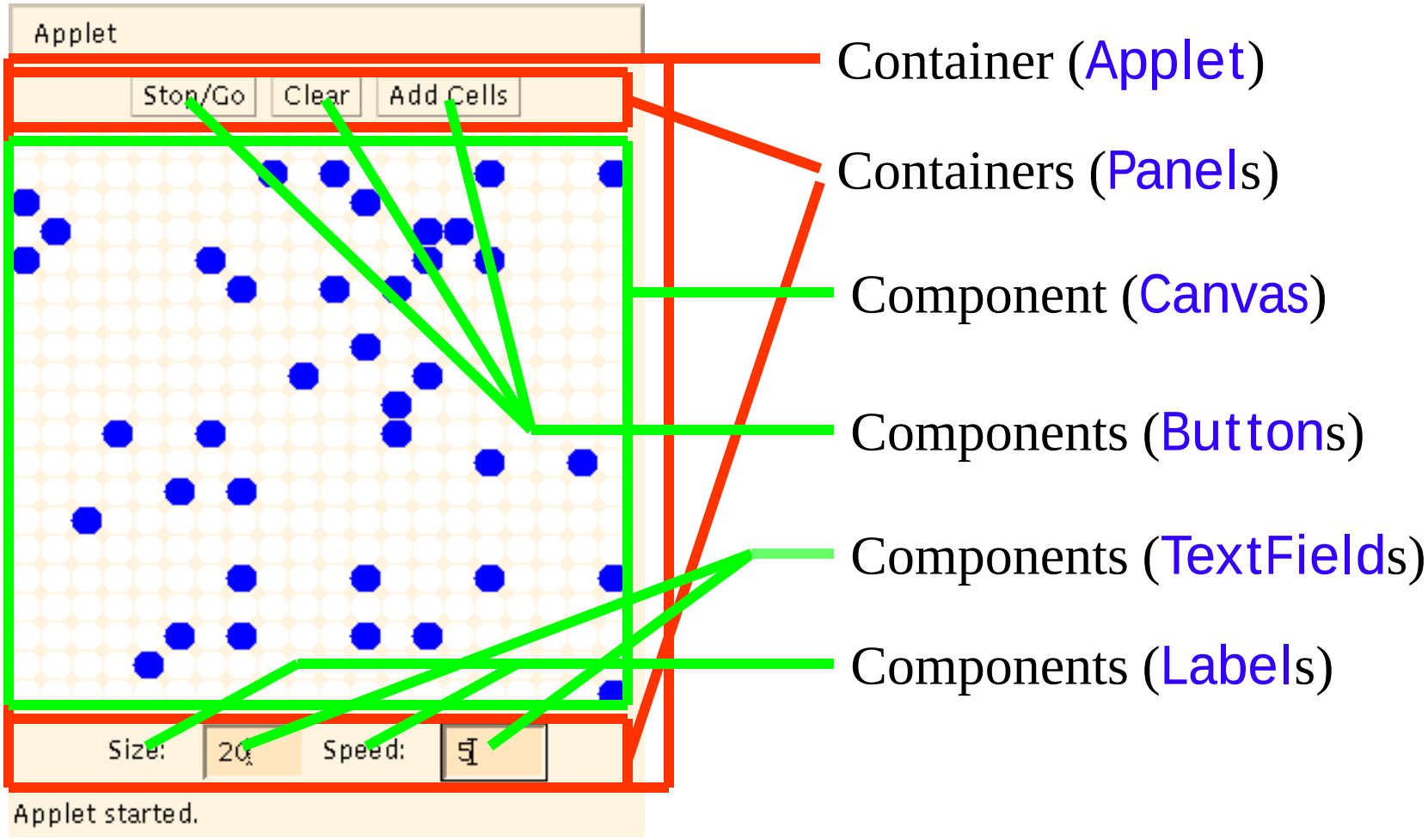
…so you can display things in an Applet

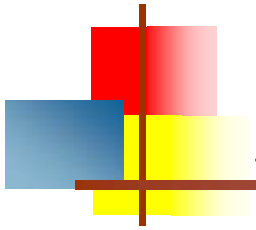
AWT Class Hierarchy



Note: There are more classes, however, these are what are covered in this chapter

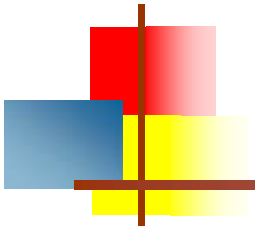
Example: A "Life" applet





Applets

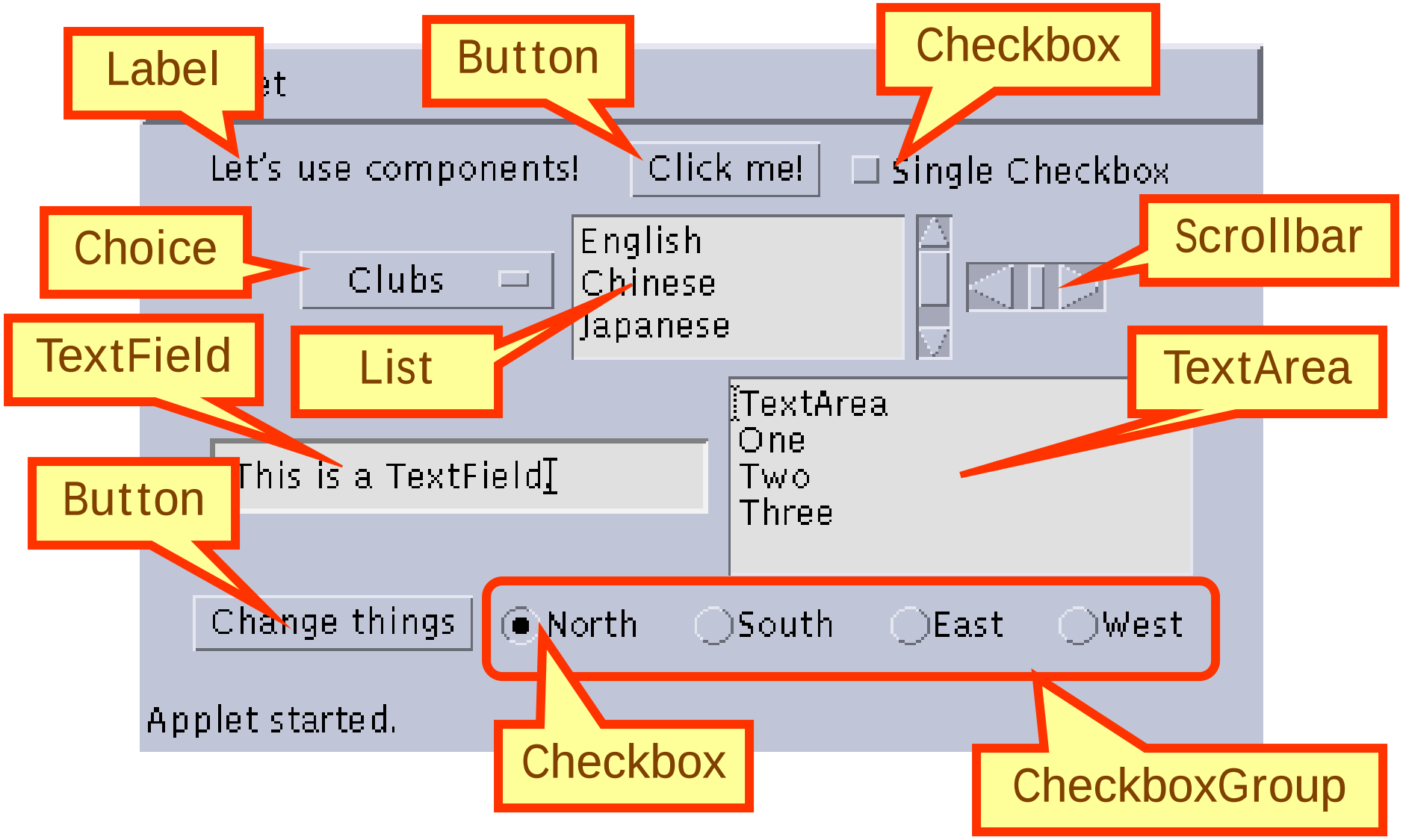
- An application has a
  `public static void main(String args[ ])` method, but an Applet usually does not
- An Applet's `main` method is in the Browser
- To write an Applet, you extend `Applet` and override some of its methods
- The most important methods are `init( )`, `start( )`, and `paint(Graphics g)`

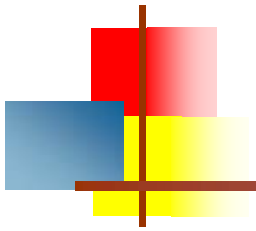


To create an applet

- public class MyApplet extends Applet { … }
  - this is the *only* way to make an Applet
- You can add components to the applet
- The best place to add components is in init( )
- You *can* paint directly on the applet, but…
- …it's better to paint on a contained component
- Do all painting from paint(Graphics g)

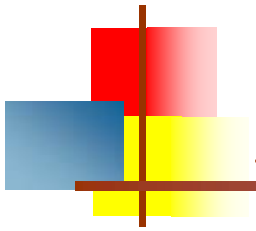
Some types of components





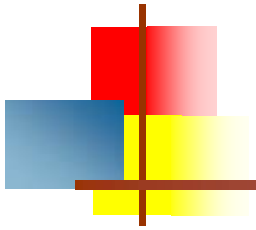
Creating components

```
Label lab = new Label ("Hi, Dave!");
Button but = new Button ("Click me!");
Checkbox toggle = new Checkbox ("toggle");
TextField txt =
    new TextField ("Initial text.", 20);
Scrollbar scrolly = new Scrollbar
   (Scrollbar.HORIZONTAL, initialValue,
    bubbleSize, minValue, maxValue);
```



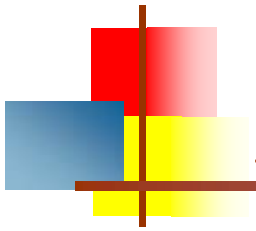
Adding components to the Applet

```
class MyApplet extends Applet {
    public void init () {
        add (lab); // same as this.add(lab)
        add (but);
        add (toggle);
        add (txt);
        add (scrolly);
        …
```



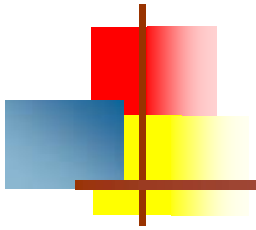
Creating a Frame

- When you create an Applet, you get a Panel "for free"
- When you write a GUI for an *application,* you need to create and use a Frame:
  - Frame frame = new Frame();
  - frame.setTitle("My Frame");
  - frame.setSize(300, 200); // width, height
  - ***... add components ...***
  - frame.setVisible(true);
- Or:
  - class MyClass extends Frame {
  - …
    setTitle("My Frame"); // in some instance method



Arranging components

- Every Container has a layout manager
- The default layout for a Panel is FlowLayout
- An Applet is a Panel
- Therefore, the default layout for a Applet is FlowLayout
- You could set it explicitly with
  `setLayout (new FlowLayout( ));`
- You could change it to some other layout manager
- The default layout for a frame is BorderLayout.



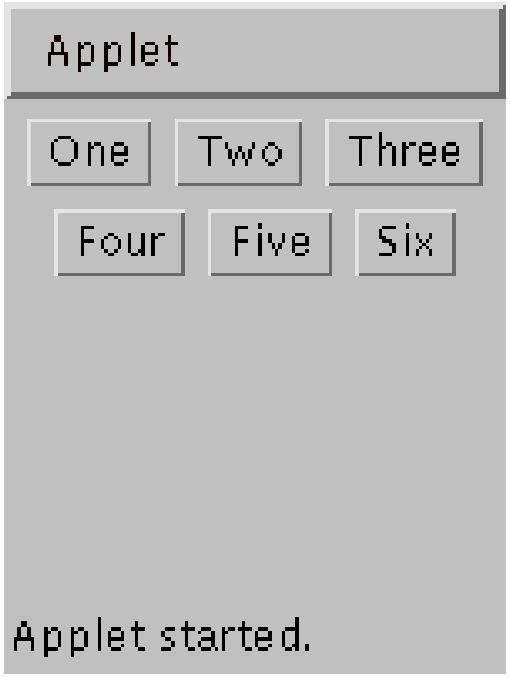
FlowLayout

- Use `add(`*component*`);` to add to a component when using a `FlowLayout`
- Components are added left-to-right
- If no room, a new row is started
- Exact layout depends on size of Applet
- Components are made as small as possible
- `FlowLayout` is convenient but often ugly

Complete example: FlowLayout

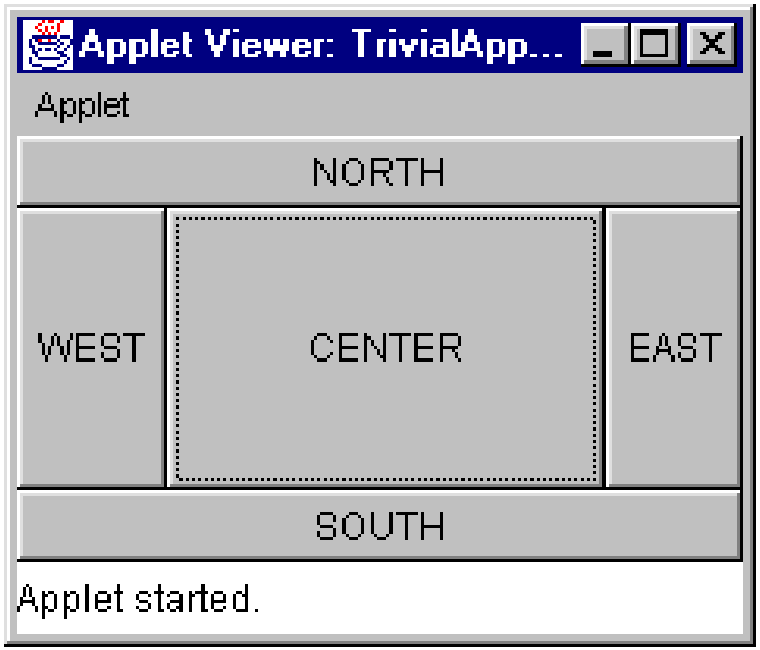
```
import java.awt.*;
import java.applet.*;

public class FlowLayoutExample extends Applet {
  public void init () {
    setLayout (new FlowLayout ()); // default
    add (new Button ("One"));
    add (new Button ("Two"));
    add (new Button ("Three"));
    add (new Button ("Four"));
    add (new Button ("Five"));
    add (new Button ("Six"));
  }
}
```

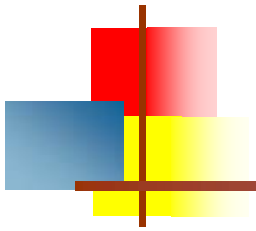


BorderLayout

- At most five components can be added
- If you want more components, add a Panel, then add components to it.
- setLayout (new BorderLayout());




```
add (new Button("NORTH"), BorderLayout.NORTH);
```



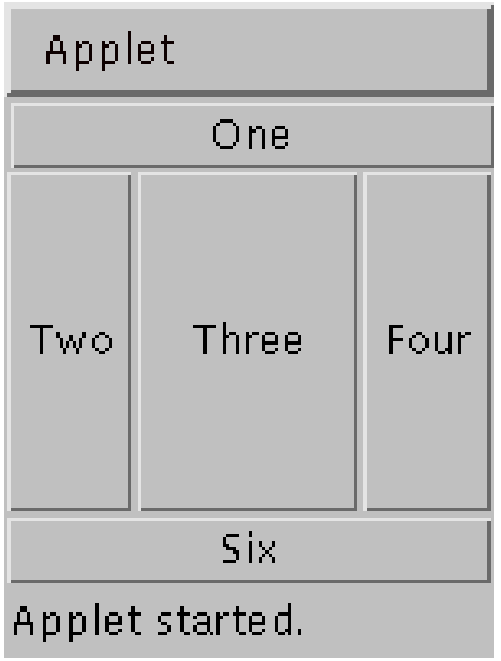
BorderLayout with five Buttons

```
public void init() {
   setLayout (new BorderLayout ());
   add (new Button ("NORTH"), BorderLayout.NORTH);
   add (new Button ("SOUTH"), BorderLayout.SOUTH);
   add (new Button ("EAST"), BorderLayout.EAST);
   add (new Button ("WEST"), BorderLayout.WEST);
   add (new Button ("CENTER"), BorderLayout.CENTER);
}
```

Complete example: BorderLayout

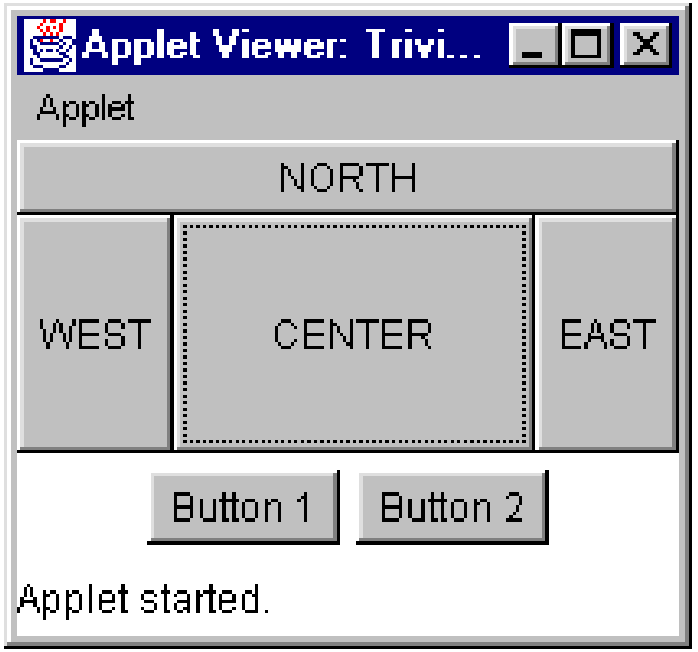
```
import java.awt.*;
import java.applet.*;

public class BorderLayoutExample extends Applet {
    public void init () {
        setLayout (new BorderLayout());
        add(new Button("One"), BorderLayout.NORTH);
        add(new Button("Two"), BorderLayout.WEST);
        add(new Button("Three"), BorderLayout.CENTER);
        add(new Button("Four"), BorderLayout.EAST);
        add(new Button("Five"), BorderLayout.SOUTH);
        add(new Button("Six"), BorderLayout.SOUTH);
    }
}
```



Using a Panel

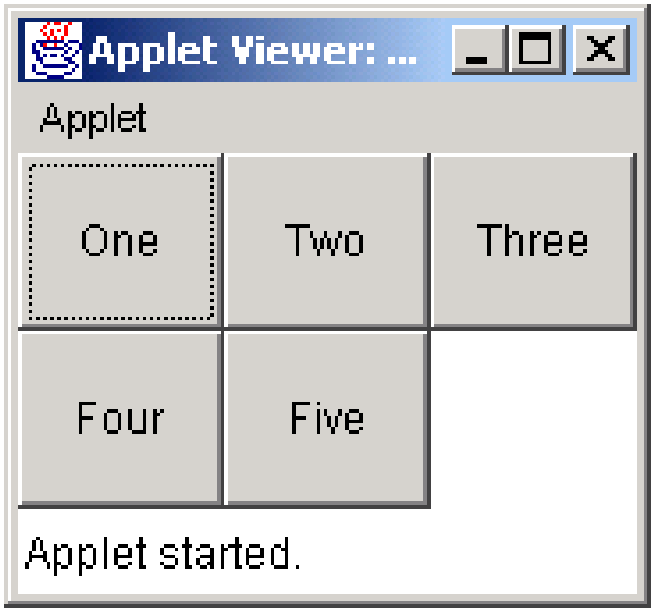
```
Panel p = new Panel();
add (p, BorderLayout.SOUTH);
p.add (new Button ("Button 1"));
p.add (new Button ("Button 2"));
```



GridLayout

- The GridLayout manager divides the container up into a given number of rows and columns:

  new GridLayout(*rows, columns*)

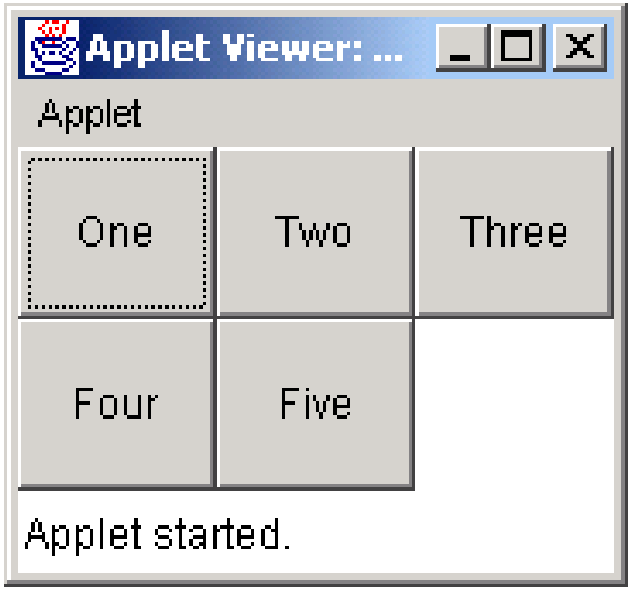


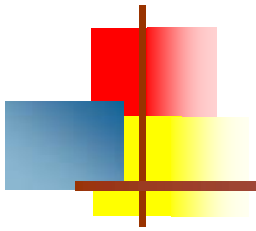
- All sections of the grid are equally sized and as large as possible

Complete example: GridLayout

```
import java.awt.*;
import java.applet.*;

public class GridLayoutExample extends Applet {
  public void init () {
    setLayout(new GridLayout(2, 3));
    add(new Button("One"));
    add(new Button("Two"));
    add(new Button("Three"));
    add(new Button("Four"));
    add(new Button("Five"));
  }
}
```

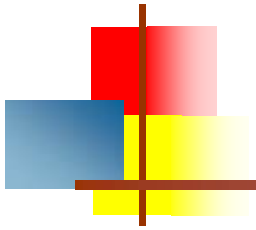




What if I dont want a LayoutManager?

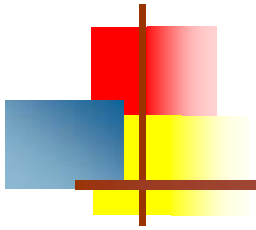
- LayoutManagers have proved to be difficult and frustrating to deal with.
- The LayoutManager can be removed from a Container by invoking its setLayout method with a null parameter.

```
Panel aPanel = new Panel();
aPanel.setLayout(null);
```



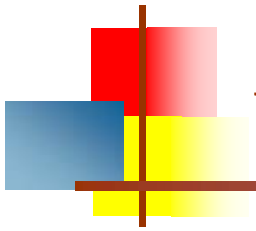
Making components active

- Most components already *appear* to do something--buttons click, text appears
- To associate an action with a component, attach a *listener* to it
- Components send events, listeners listen for events
- Different components may send different events, and require different listeners



Listeners

- Listeners are interfaces, not classes
  - class MyButtonListener implements ActionListener {
- An interface is a group of methods that *must* be supplied
- When you say implements, you are *promising* to supply those methods



Writing a Listener

- For a Button, you need an ActionListener

```
b1.addActionListener
      (new MyButtonListener ( ));
```

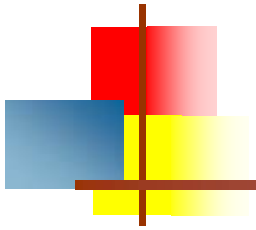
- An ActionListener must have an actionPerformed(ActionEvent) method

```
public void actionPerformed(ActionEvent e) {
    ...
}
```

MyButtonListener


```
public void init () {
   ...
   b1.addActionListener (new MyButtonListener ());
}

 class MyButtonListener implements ActionListener {
   public void actionPerformed (ActionEvent e) {
     showStatus ("Ouch!");
   }
 }
```



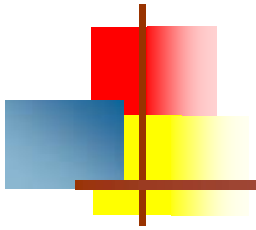
Listeners for TextFields

- An ActionListener listens for someone hitting the Enter key
- An ActionListener requires this method:

  public void actionPerformed (ActionEvent e)
- You can use getText( ) to get the text

- A TextListener listens for any and all keys
- A TextListener requires this method:

  public void textValueChanged(TextEvent e)

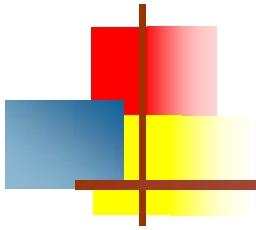


Event Sources and Their Listeners

- Dialog - WindowListener
- Frame - WindowListener
- Button - ActionListener
- Choice - ItemListener
- Checkbox – ItemListener
- List - ItemListener, ActionListener (when an item is double clicked)
- Scrollbar - AdjustmentListener
- TextField - ActionListener, TextListener
- TextArea - TextListener

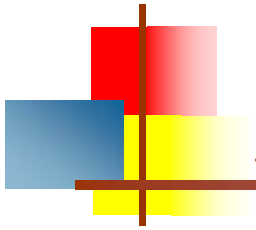
Simple AWT Example

```
import java.awt.*;
import java.awt.event.*;
public class SimpleAWT extends java.applet.Applet
implements ActionListener, ItemListener {
private Button button = new Button("Push Me!");
private Checkbox checkbox = new Checkbox("Check Me!");
private Choice choice = new Choice();
private Label label = new Label("Pick something!");
public void init() {
button.addActionListener(this);
checkbox.addItemListener(this);
choice.addItemListener(this);
// An Applet is a Container because it extends Panel.
setLayout(new BorderLayout());
choice.addItem("Red");
choice.addItem("Green");
choice.addItem("Blue");
Panel panel = new Panel();
panel.add(button);
panel.add(checkbox);
panel.add(choice);
add(label, "Center");
add(panel, "South");
}
```



Simple AWT Example Cont.

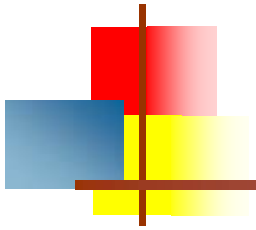
```
public void actionPerformed(ActionEvent e) {
if (e.getSource() == button) {
label.setText("The Button was pushed.");
}
}
public void itemStateChanged(ItemEvent e) {
if (e.getSource() == checkbox) {
label.setText("The Checkbox is now " +
checkbox.getState() + ".");
} else if (e.getSource() == choice) {
label.setText(choice.getSelectedItem() + " was selected.");
}
}
}
```



AWT and Swing

- AWT Buttons vs. Swing JButtons:
  - A Button is a Component
  - A JButton is an AbstractButton, which is a JComponent, which is a Container, which is a Component
- Containers:
  - Swing uses AWT Containers
- AWT Frames vs. Swing JFrames:
  - A Frame is a Window is a Container is a Component
  - A JFrame is a Frame, etc.
- Layout managers:
  - Swing uses the AWT layout managers, plus a couple of its own
- Listeners:
  - Swing uses many of the AWT listeners, plus a couple of its own

- Bottom line: Not only is there a lot of similarity between AWT and Swing, but Swing actually uses much of the AWT



Graphics

- It is possible to draw lines and various shapes within a Panel under the AWT.
- Each Component contains a Graphics object which defines a Graphics Context which can be obtained by a call to getGraphics().
- Common methods used in Graphics include:

drawLine
drawOval
drawRect
drawRoundRect
drawArc

fillArc
fillOval
fillRect
fillRoundRect
setColor
setFont
drawImage
drawString

Java Graphics

To start, here's a basic applet that demonstrates Java graphics using AWT:

```
 import java.awt.*;
import java.applet.Applet;

public class BasicGraphics extends Applet {
   public void paint(Graphics g) {
      g.setColor(Color.red);
      g.fillRect(10, 20, 40, 40);
   }  // end paint()
} // end class BasicGraphics
```

Try to compile this code and run it as an applet.

What do you see?

Java Graphics: The Core

```
import java.awt.*;
    import java.applet.Applet;

    public class BasicGraphics extends
    Applet {
      public void paint(Graphics g) {
        g.setColor(Color.red);
        g.fillRect(10, 20, 40, 40);
      }  // end paint()
    } // end class BasicGraphics
```

The first lines are import statements, to load the AWT and Applet libraries.

If you were making a Swing version, you would load Swing libraries.

Java Graphics: The Core

```
import java.awt.*;
    import java.applet.Applet;

    public class BasicGraphics extends
    Applet {
      public void paint(Graphics g) {
        g.setColor(Color.red);
        g.fillRect(10, 20, 40, 40);
      }  // end paint()
    } // end class BasicGraphics
```

Our class is called BasicGraphics, and it extends Applet to inherit Applet properties.

Java Graphics: The Core

```
import java.awt.*;
    import java.applet.Applet;

    public class BasicGraphics extends Applet {
      public void paint(Graphics g) {
        g.setColor(Color.red);
        g.fillRect(10, 20, 40, 40);
      }  // end paint()
    } // end class BasicGraphics
```

Inside the applet, we have just one method: paint()

(Remember from Applets that other methods are optional.)

It has one parameter, called the "abstract Graphics object", and we call it "g".

Get used to this, you need to tell paint() what to paint on!

Java Graphics: The Core

```
import java.awt.*;
    import java.applet.Applet;

    public class BasicGraphics extends Applet {
      public void paint(Graphics g) {
        g.setColor(Color.red);
        g.fillRect(10, 20, 40, 40);
      }  // end paint()
    } // end class BasicGraphics
```

paint() has two methods inside of it: setColor and fillRect

g.setColor(Color.red); is the command to color whatever graphic "thing" we have "red".

The computer still doesn't know what Graphic "thing" g is going to be!

Java Graphics: The Core

```
import java.awt.*;
    import java.applet.Applet;

    public class BasicGraphics extends Applet {
      public void paint(Graphics g) {
        g.setColor(Color.red);
        g.fillRect(10, 20, 40, 40);
      }  // end paint()
    } // end class BasicGraphics
```

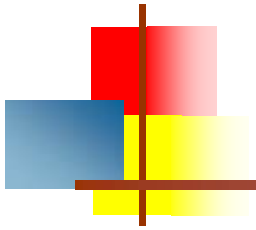
g.fillRect(10, 20, 40, 40) tells the applet to make g into a "fillRect", which is a "filled rectangle".

10 is the starting x-position in the applet,

20 is the starting y-position in the applet

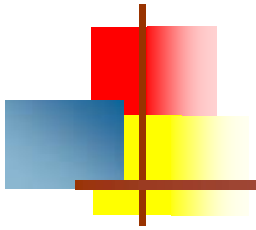
40 is the width and the other 40 is the height.

Remember what color will fill it? Red!



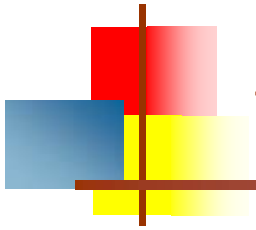
Summary I: Building a GUI

- Create a container, such as Frame or Applet
- Choose a layout manager
- Create more complex layouts by adding Panels; each Panel can have its own layout manager
- Create other components and add them to whichever Panels you like



Summary II: Building a GUI

- For each active component, look up what kind of Listeners it can have
- Create (implement) the Listeners
  - often there is one Listener for each active component
  - Active components can share the same Listener
- For each Listener you implement, supply the methods that it requires
- For Applets, write the necessary HTML



Vocabulary

- AWT – The Abstract Window Toolkit provides basic graphics tools (tools for putting information on the screen)
- Swing – A much better set of graphics tools
- Container – a graphic element that can hold other graphic elements (and is itself a Component)
- Component – a graphic element (such as a Button or a TextArea) provided by a graphics toolkit
- listener – A piece of code that is activated when a particular kind of event occurs
- layout manager – An object whose job it is to arrange Components in a Container