Java

GUI building with the AWT

AWT (Abstract Window Toolkit)

- Present in all Java implementations
- Described in (almost) every Java textbook
- 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

- Requires Java 2 or a separate (huge) download
- More controls, and they are more flexible
- Gives a choice of "look and feel" packages
- Much easier to build an attractive GUI
- import javax.swing.*;

Swing vs. AWT

- Swing is bigger and slower
- 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
- AWT: Button b = new Button ("OK");
  Swing: JButton b = new JButton("OK");

To build a GUI...

- Make somewhere to display things -- a Frame, a Window, 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 & executes code to deal with it

Containers and Components

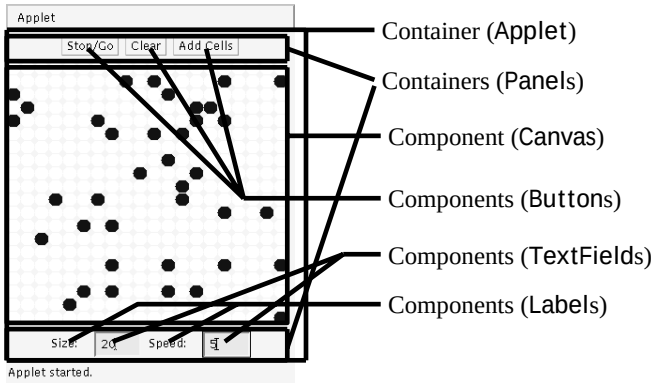
- The job of a Container is to hold and display Components
- Some common subclasses of Component are Button, Checkbox, Label, Scrollbar, TextField, **and** TextArea
- A Container is also a Component
- Some Container subclasses are Panel (and Applet), Window, and Frame

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

Example: A "Life" applet



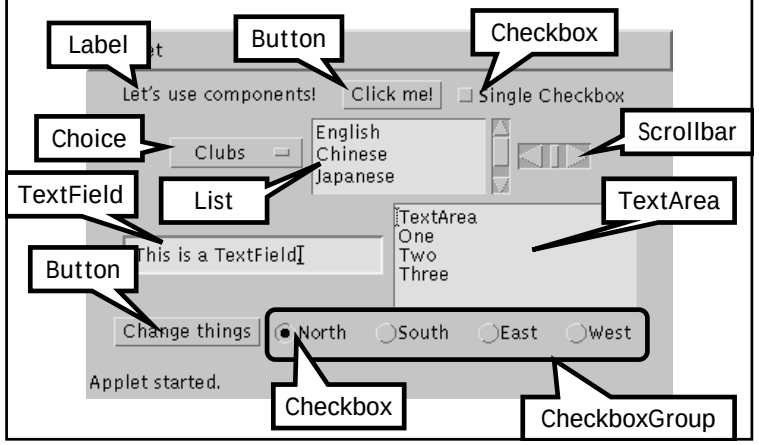
Applets

- 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!");
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);
    …
```

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 another layout manager

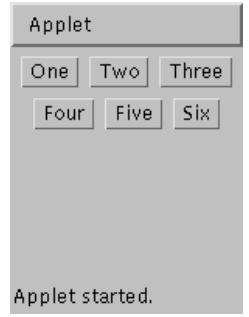
FlowLayout

- 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

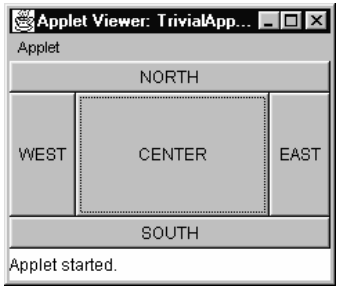
```
import java.awt.*;
import java.applet.*;

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



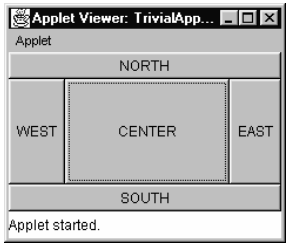
BorderLayout

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




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

BorderLayout with five Buttons

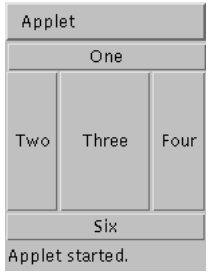



```
public void init() {
  setLayout (new BorderLayout ());
  add (BorderLayout.NORTH, new Button ("NORTH"));
  add (BorderLayout.SOUTH, new Button ("SOUTH"));
  add (BorderLayout.EAST, new Button ("EAST"));
  add (BorderLayout.WEST, new Button ("WEST"));
  add (BorderLayout.CENTER,new Button("CENTER"));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

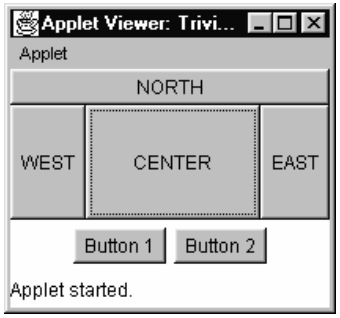
Complete example: BorderLayout

```
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);
  }
}
```



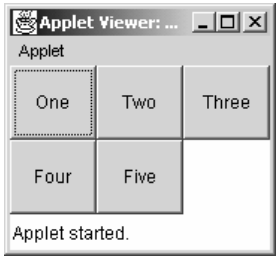
Using a Panel




```
Panel p = new Panel();
add (BorderLayout.SOUTH, p);
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:



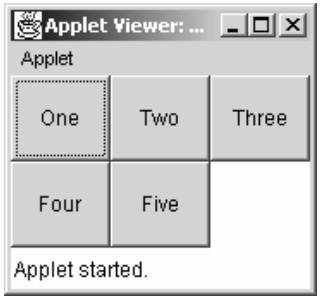
new GridLayout(*rows, columns*)

- 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"));
  }
}
```



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

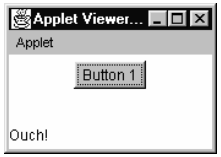
- For a Button, you need an ActionListener

```
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

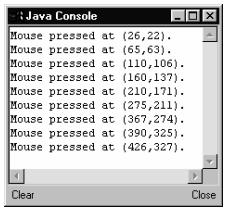
- A TextListener listens for any and all keys
- A TextListener requires this method:
  public void textValueChanged(TextEvent e)

Example: Mouse CLicks

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

public class ClickReporter extends Applet {
  public void init() {
    setBackground(Color.yellow);
    addMouseListener(new ClickListener());
  }
}
```

Mouse Clicks




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

public class ClickListener
  extends MouseAdapter {
  public void mousePressed(MouseEvent event) {
    System.out.println(
        "Mouse pressed at (" +
        event.getX() +
        "," +
        event.getY() +
        ")." );
  }
}
```

Summary: Standard AWT Event Listeners

Listener	Adapter Class (If Any)	Registration Method
ActionListener		addActionListener
AdjustmentListener		addAdjustmentListener
ComponentListener	ComponentAdapter	addComponentListener
ContainerListener	ContainerAdapter	addContainerListener
FocusListener	FocusAdapter	addFocusListener
ItemListener		addItemListener
KeyListener	KeyAdapter	addKeyListener
MouseListener	MouseAdapter	addMouseListener
MouseMotionListener	MouseMotionAdapter	addMouseMotionListener
TextListener		addTextListener
WindowListener	WindowAdapter	addWindowListener

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 I

- 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

Vocabulary II

- 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