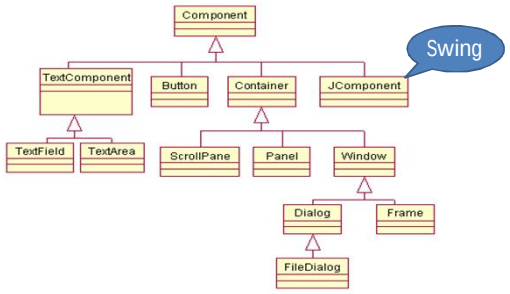
Java

GUI building with the AWT

AWT (Abstract Window Toolkit)

- Present in all Java implementations
- 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.*;`

General Layout of AWT



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

Top Level Windows

- **Window:** A top-level window that has no border.
- **Frame:** A top-level window that has a border and can also have an associated MenuBar.
- **Dialog:** A top-level window used to create dialogs. One subclass of this is the FileDialog.
- **Panel:** Subclass of container used inside other containers.
  - Used to store collections of objects.
  - Does not create a separate window of its own.

Important I/O Components

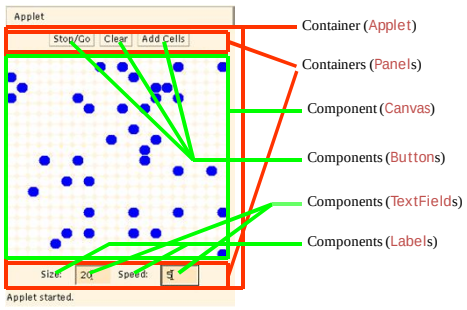
- **Button:** A push button.
- **Canvas:** General-purpose component that lets you paint or trap input events from the user. It can be used to create graphics.
- **Checkbox:** A component that has an "on" or "off" state. You can place Checkboxes in a group that allows at most 1 box to be checked.
- **Choice:** A component that allows the selection of one of a group of choices. Takes up little screen space.
- **Label:** A component that displays a static string at a certain location.
- **List:** A scrolling list of strings. Allows one or several items to be selected.
- **TextArea:** Multiple line text components that allows viewing and editing.
- **TextField:** A single-line text component that can be used to build forms.

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



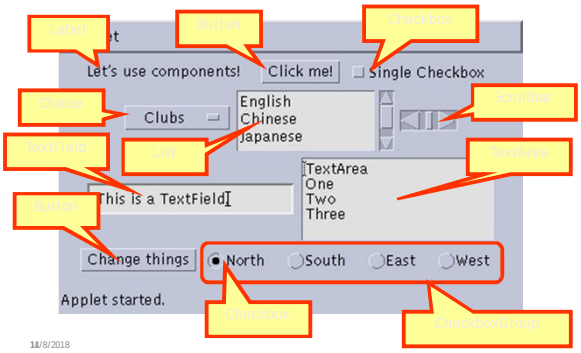
Applets

- 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, Dear!");
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

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

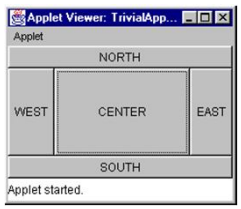
Complete example: FlowLayout

```
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);
}
```

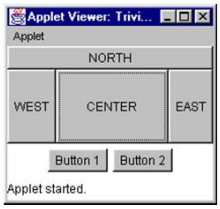

Complete example: BorderLayout

```
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

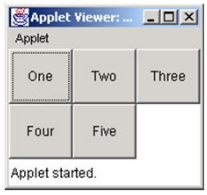
```
Panel p = new Panel();
add (p, BorderLayout.SOUTH);
p.add (new Button ("Button 1"));
p.add (new Button ("Button 2"));
```



GridLayout

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

  new GridLayout(*rows*, *columns*)

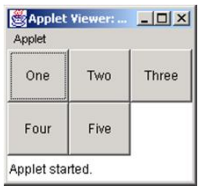


- All sections of the grid are equally sized and as large as possible

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"));
  }
}
```



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)

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

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

The End