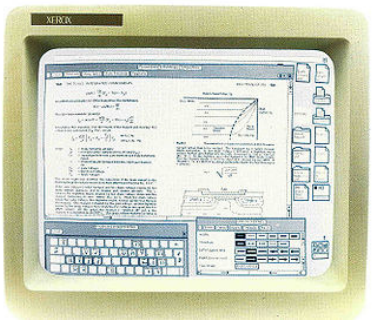
cs2220: Engineering Software

Class 22: Graphical User Interfaces



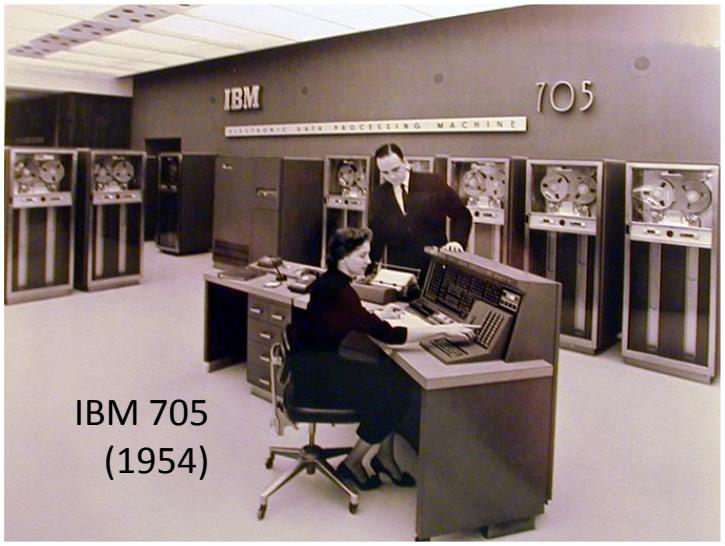
Xerox Star

Fall 2010
UVa
David Evans

Plan for Today

- History of Interactive Computing
- Building GUIs in Java

Design Reviews this week!



Computer as "Clerk": Augmenting Human Intellect

In such a future working relationship between human problem-solver and computer 'clerk,' the capability of the computer for executing mathematical processes would be used whenever it was needed. However, the computer has many other capabilities for manipulating and displaying information that can be of significant benefit to the human in nonmathematical processes of planning, organizing, studying, etc. Every person who does his thinking with symbolized concepts (whether in the form of the English language, pictographs, formal logic, or mathematics) should be able to benefit significantly.

Douglas Engelbart, *Augmenting Human Intellect* (1962)

Engelbart's Demo (1968)

- First Mouse
- Papers and folders
- Videoconferencing
- Email
- Hypertext
- Collaborative editing

monday afternoon

december 9

3:45 p.m. / arena

Chairman:
DR. D. C. ENGELBART
Stanford Research Institute
Menlo Park, California

a research center for augmenting human intellect

This session is entirely devoted to a presentation by Dr. Engelbart on a computer-based, interactive, multiconsole display system which is being developed at Stanford Research Institute under the sponsorship of ARPA, NASA and RADC. The system is being used as an experimental laboratory for investigating principles by which interactive computer aids can augment intellectual capability. The techniques which are being described will, themselves, be used to augment the presentation.
The session will use an on-line, closed circuit television hook-up to the SRI computing system in Menlo Park.
Following the presentation remote terminals to the system, the remainder of the side for that purpose.

http://www.sri.com/news/storykits/1968video.html

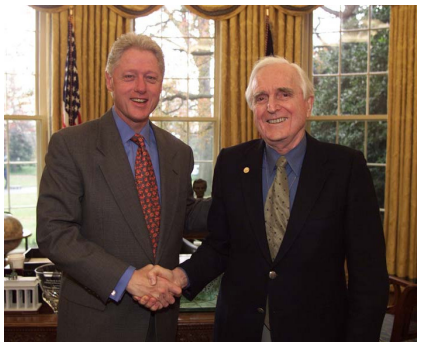
Doug Engelbart's Mouse (1968)

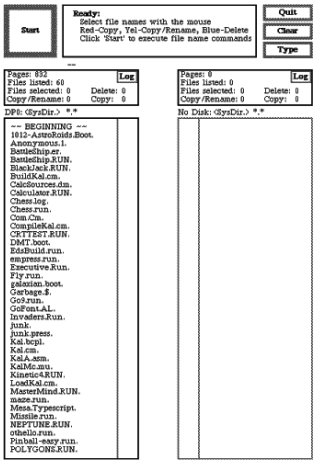
Claude Shannon, "Theseus" (1950)

"We see the quickest gains emerging from (1) giving the human the minute-by-minute services of a digital computer equipped with computer-driven cathode-ray-tube display, and (2) **developing the new methods of thinking and working that allow the human to capitalize upon the computer's help**. By this same strategy, we recommend that an initial research effort develop a prototype system of this sort aimed at increasing human effectiveness in the task of computer programming."

Douglas Engelbart, *Augmenting Human Intellect* (1962)

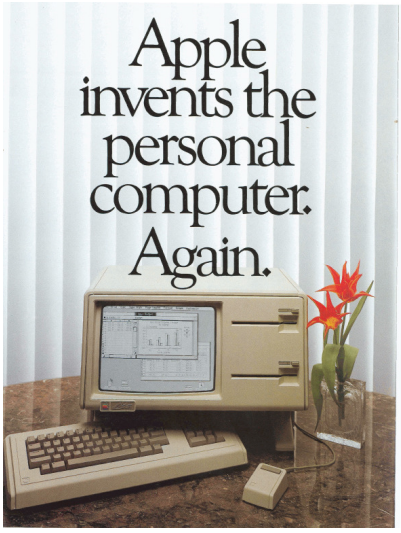
Medal of Technology 2000





Xerox Alto

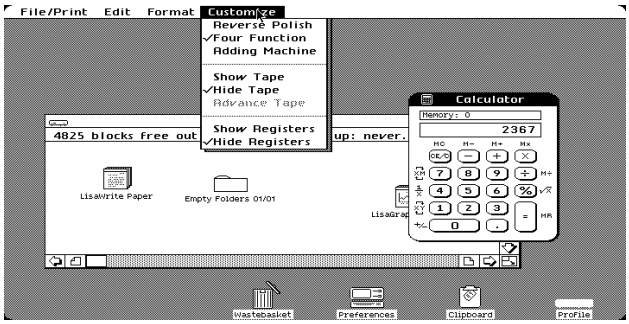
Xerox PARC, 1973



Apple Lisa

1983

Lisa Interface



http://www.guidebookgallery.org/screenshots/lisaos10

Any real progress since then?

Mac OSX

Nintendo Wii

Microsoft Kinect

Microsoft Surface

Designing GUIs

- Requires lots of skill
- Psychology, Cognitive Science
- User studies
- **Good taste**

Read Donald Norman's and Ed Tufte's books

Look at what SIS does and do the opposite!

Building GUIs

- Like all Programming
  - Encapsulation, Abstraction, Specification
  - Testing: especially hard
- Unique-ish Aspects
  - Event-Driven (network programming also like this)
  - Multi-Threaded (network, others)
  - Huge APIs

Model-View-Controller

Invented at PARC in 1970s (Smalltalk)

Model: domain data and logic

View: presents model

Controller: receives input and alters model

Goal: **abstraction**

separate display from model

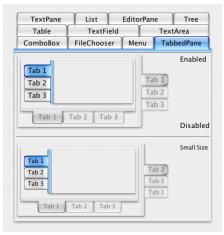
separate control interface

Java GUI Toolkits

AWT

Abstract Window Toolkit

Looks like Java

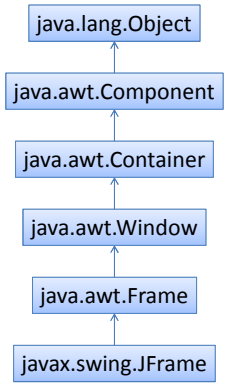


Swing

(since JDK 1.2)

real reason for Swing coming later...

Frames



Main windows are **JFrame** objects

```
JFrame frame = new JFrame("Swing GUI");
```

Window Title

JFrame Methods

```
// inherited from java.awt.Window
public void pack()
  MODIFIES: this
  EFFECTS: Causes this Window to be sized to fit the
   preferred size and layouts of its subcomponents.

// inherited from java.awt.Component
public void setVisible(boolean b)
   MODIFIES: this, display
   EFFECTS: If b, shows this.  Otherwise, hides this.
```

Swing Application

```
import javax.swing.*;

public class Main {
  private static void showGUI() {
     //Create and set up the window.
     JFrame frame = new JFrame("Swing GUI");
     frame.pack();
     frame.setVisible(true);
   }

   public static void main(String args[]) {
      javax.swing.SwingUtilities.invokeLater(new Runnable() {
         public void run() { showGUI(); }
      });
   }
}
```

Based on Sun's Swing tutorials:
http://java.sun.com/docs/books/tutorial/uiswing/learn/example1.html

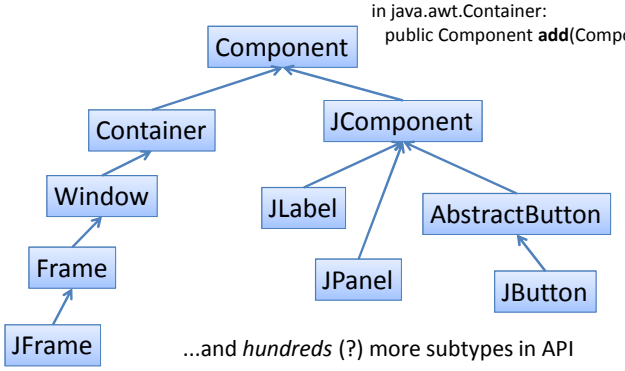
Adding to a Frame

public java.awt.Container **getContentPane**()
  EFFECTS: Returns the contentPane object for this.

in java.awt.Containter:
public Component **add**(Component c)
  MODIFIES: this
  EFFECTS: Appends *c* to the end of this container.

What can you add?

in java.awt.Container:
public Component **add**(Component c)



...and *hundreds* (?) more subtypes in API

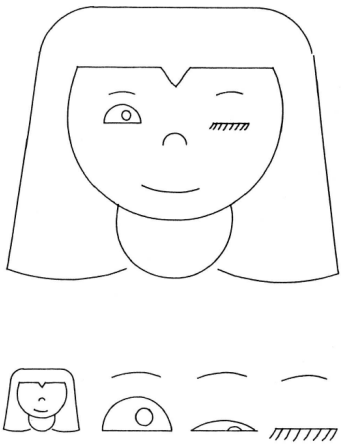
GUIs and Subtyping

In the process of making the Sketchpad system operate, a few very general functions were developed which make no reference at all to the specific types of entities on which they operate. These general functions give the Sketchpad system the ability to operate on a wide range of problems. The motivation for making the functions as general as possible came from the desire to get as much result as possible from the programming effort involved. For example, the general function for expanding instances makes it possible for Sketchpad to handle *any* fixed geometry subpicture. The rewards that come from implementing general functions are so great that the author has become reluctant to write any programs for specific jobs.

Each of the general functions implemented in the Sketchpad system abstracts, in some sense, some common property of pictures independent of the specific subject matter of the pictures themselves.

Ivan Sutherland, *Sketchpad: a Man-Machine Graphical Communication System*, 1963 (major influence on Alan Kay inventing OOP in 1970s)

Components in Sketchpad



Adding Components

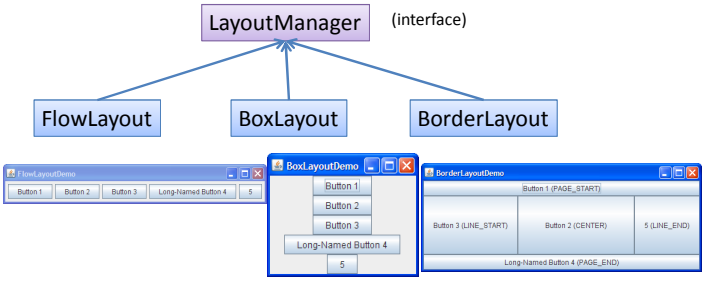
```
import javax.swing.*;

public class Main {
  private static void showGUI() {
     //Create and set up the window.
     JFrame frame = new JFrame("Swing GUI");
     java.awt.Container content = frame.getContentPane();
     content.add(new JLabel ("Yo!"));
     content.add(new JButton ("Click Me"));
     frame.pack();
     frame.setVisible(true);
  }

  public static void main(String args[]) {
    ...
  }
}
```

What happened to "Yo!"?

Layout

// in Container:
public void **setLayout**(LayoutManager mgr)
  MODIFIES: this
  EFFECTS: sets the layout manager to mgr
    for this container.

LayoutManager Implementations



...about 30 more in API!

http://java.sun.com/docs/books/tutorial/uiswing/layout/visual.html

Adding Components

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

public class Main {
  private static void showGUI() {
     //Create and set up the window.
     JFrame frame = new JFrame("Swing GUI");
     java.awt.Container content = frame.getContentPane();
     content.setLayout(new FlowLayout());
     content.add(new JLabel ("Yo!"));
     content.add(new JButton ("Click Me"));
     frame.pack();
     frame.setVisible(true);
  }
```

Don’t try this at home?

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

public class Main {
  private static void showGUI() {
      //Create and set up the window.
      JFrame frame = new JFrame("Swing GUI");
      java.awt.Container content = frame.getContentPane();
      content.setLayout(new FlowLayout());
      content.add(frame);
      frame.pack();
      frame.setVisible(true);
  }
```

Exception in thread "AWT-EventQueue-0"
java.lang.IllegalArgumentException: adding container's parent to itself

Making Buttons Do Something

public void **addActionListener**(ActionListener l)
MODIFIES: this
EFFECTS: Adds an ActionListener l to the button.

java.awt.event
interface ActionListener extends EventListener {
void **actionPerformed** (ActionEvent e)
EFFECTS: anything
Note: this method is called when an action occurs.
}

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class ButtonListener implements ActionListener {
  public void actionPerformed (ActionEvent e) {
     System.out.println ("Got a button press:" + e);
  }
}

public class Main {
  private static void showGUI() {
      JFrame frame = new JFrame("Swing GUI");
      java.awt.Container content = frame.getContentPane();
      content.setLayout(new FlowLayout());
      content.add(new JLabel ("Yo!"));
      JButton button = new JButton ("Click Me");
      button.addActionListener(new ButtonListener());
      content.add(button);
      frame.pack();
      frame.setVisible(true);
   }
```

Action Events

Got a button press: java.awt.event.ActionEvent[ACTION_PERFORMED,cmd=Click Me, when=1163559916342,modifiers=Button1] on javax.swing.JButton[,27,5,82x26,alignmentX=0.0,alignmentY=0.5, border=javax.swing.plaf.BorderUIResource$CompoundBorderUIResource@29ab3e ,flags=296,maximumSize=,minimumSize=, preferredSize=,defaultIcon=,disabledIcon=,disabledSelectedIcon=, margin=javax.swing.plaf.InsetsUIResource[top=2,left=14,bottom=2,right=14],paint Border=true,paintFocus=true, pressedIcon=,rolloverEnabled=true,rolloverIcon=,rolloverSelectedIcon=, selectedIcon=,text=Click Me,defaultCapable=true]

```
class ButtonListener implements ActionListener {
  public void actionPerformed (ActionEvent e) {
    if (e.getActionCommand().equals ("On")) {
      System.out.println("On!");
    } else if (e.getActionCommand().equals("Off")) {
      System.out.println("Off!");
    } else {
      System.out.println("Unrecognized button press!");
    }
  }
}

public class Main {
  private static void showGUI() {
      …
      ButtonListener bl = new ButtonListener();
      JButton onButton = new JButton ("On");
      onButton.addActionListener(bl);
      content.add(onButton);
      JButton offButton = new JButton ("Off");
      offButton.addActionListener(bl);
      content.add(offButton);
      …
```

Activating/Deactivating

// in JButton:
void setEnabled(boolean b)
MODIFIES: this
EFFECTS: If b, enables this.
Otherwise, disables this.

```
class ButtonListener implements ActionListener {
  public void actionPerformed (ActionEvent e) {
    if (e.getActionCommand().equals ("On")) {
      System.out.println("On!");
    } else if (e.getActionCommand().equals("Off")) {
      System.out.println("Off!");
    } else {
      System.out.println("Unrecognized button press!"); } } }

public class Main {
  private static void showGUI() {
    ...
    ButtonListener bl = new ButtonListener();
    JButton onButton = new JButton ("On");
    onButton.addActionListener(bl);
    content.add(onButton);
    JButton offButton = new JButton ("Off");
    offButton.addActionListener(bl);
    content.add(offButton);
    ...
```

Can we make clicking "On" enable the "Off" button (and vice versa)?

Inner Classes

- Added to JDK 1.1 (no JavaVM support)
- Define a **class** inside a scope
- It has access to variables in the containing scope including **private** instance variables!

What deficiency in Java is this making up for?

No **lambda**! There is no way in Java to dynamically construct a procedure. Inner classes provide a more cumbersome, less expressive (but easier to typecheck statically) substitute.

```
public class Main {
  private static void showGUI() {
    JFrame frame = new JFrame("Swing GUI");
    java.awt.Container content = frame.getContentPane();
    content.setLayout(new FlowLayout());
    final JButton onButton = new JButton ("On");
    final JButton offButton = new JButton ("Off");

    class ButtonListener implements ActionListener {
      public void actionPerformed (ActionEvent e) {
        if (e.getActionCommand().equals ("On")) {
          onButton.setEnabled(false);
          offButton.setEnabled(true);
        } else if (e.getActionCommand().equals("Off")) {
          onButton.setEnabled(true);
          offButton.setEnabled(false);
        }
      }
    };
    ButtonListener bl = new ButtonListener();
    onButton.addActionListener(bl);
    content.add(onButton);
```

Anonymous Classes

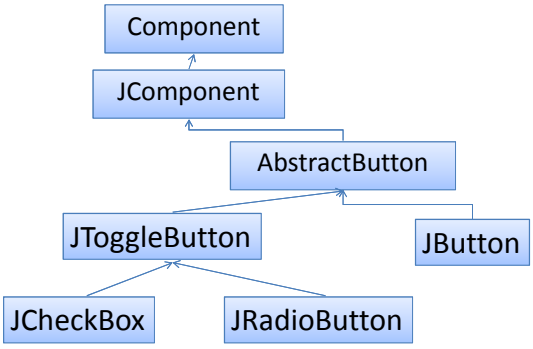
No need to give inner classes names!

var = new *Superclass* ( ) {
  // override methods here
}

```
public class Main {
  private static void showGUI() {
    JFrame frame = new JFrame("Swing GUI");
    java.awt.Container content = frame.getContentPane();
    content.setLayout(new FlowLayout());
    final JButton onButton = new JButton ("On");
    final JButton offButton = new JButton ("Off");

    ActionListener bl = new ActionListener () {
      public void actionPerformed (ActionEvent e) {
        if (e.getActionCommand().equals ("On")) {
          onButton.setEnabled(false);
          offButton.setEnabled(true);
        } else if (e.getActionCommand().equals("Off")) {
          onButton.setEnabled(true);
          offButton.setEnabled(false);
        }
      }
    };
    onButton.addActionListener(bl);
    content.add(onButton);
```

What is the actual type of **bl**?

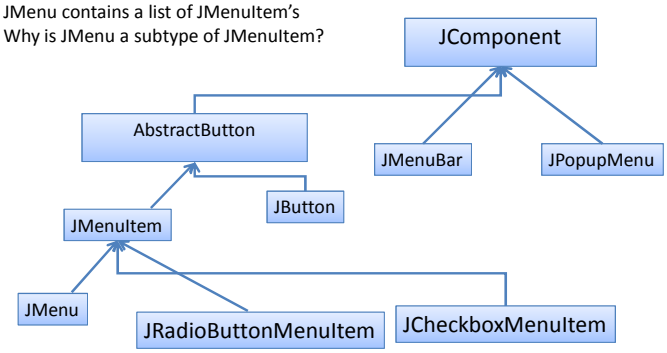
Not just Buttons



http://java.sun.com/docs/books/tutorial/uiswing/components/button.html

Menus Too...

Awkward design:
JMenu is a button – action is to popup
JPopupMenu
JMenu contains a list of JMenuItem's
Why is JMenu a subtype of JMenuItem?



http://java.sun.com/docs/books/tutorial/uiswing/components/menu.html

Concurrency in GUIs

- Responsiveness of GUI depends on multiple threads
- Swing thread types:
  - *Initial threads* (start program)
  - *One event dispatch thread* (all event-handling code)
  - *Worker threads* (do time-consuming tasks in background)

Swing framework does most of the work – programmer doesn't need to create threads

Event Dispatch

Why is there only one event dispatch thread?

Hint: did we need to synchronize?

One event thread means all compute-intensive work should be done in worker threads. (Otherwise interface freezes like ps4 ImageChop).

Worker Threads

class javax.swing.SwingWorker<T,V>

Create a background thread to do compute-intensive tasks

http://download.oracle.com/javase/6/docs/api/javax/swing/SwingWorker.html
(added to JDK 1.6)

Charge

- GUI APIs are subtyping and inheritance paradises, concurrency morasses
- GUI APIs are huge and complex
  - Java's is especially complex because of AWT + Swing, and portability

Creating a *simpler* GUI requires *more complex* programming