Original GUIs
current GUIs

Artist
Rene Magritte
Belgium surrealist 1898-1967

Author
G. Michael Barnes
Computer Science Dept
California State University Northridge
www.csun.edu/~renzo
818.677.2299

This is not a pipe
Why GUIs?

Computer Graphics technology enabled GUIs and computer gaming.

GUI's were 1985 breakout computer technology.

Without a GUI there would be not be any, or much less:

- Computer based industry – USA's major productivity advantage
- Computer systems in the homes and schools
- Internet, Web, E-commerce
  - before Mosaic – telnet, ftp, gopher, WAIS
  - after Mosaic – AOL, Amazon.com, Facebook, Link-in
- smart phones, tablets

How much software would you use / buy that does not have a GUI interface?

How many computer games without graphics would you play?

Where would Microsoft be without Windows? Apple w/o Macintosh?
Course Objectives

Experience with 2+ GUI APIs – Java Swing & Microsoft Windows Presentation Foundation (WPF)

On-event software architecture – event listeners / handlers

Comparison and evaluation of relatively large class libraries (GUI)

Experience with Interface Builders
    designer’s that “draw your interface”
    scripting languages that “build your interface” – XAML

Discussion, evaluation, of effective GUI design concepts / tradeoffs
Z order

With apologies to Rene Magritte

About

Ceci n'est pas une pipe

JMenu
JLabel
JDialog

events

JButton

JFrame
<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1963</td>
<td>SketchPad</td>
<td>interactive computer graphics system</td>
</tr>
<tr>
<td>1968</td>
<td>NLS, augment</td>
<td>2D GUI, mouse, hypertext, windows, editing</td>
</tr>
<tr>
<td>1977</td>
<td>Apple II</td>
<td>color graphics personal computer</td>
</tr>
<tr>
<td>1981</td>
<td>Xerox Star</td>
<td>OOP, bit mapped GUI, ethernet, smalltalk</td>
</tr>
<tr>
<td>1984</td>
<td>Macintosh</td>
<td>personal GUI</td>
</tr>
<tr>
<td>1984</td>
<td>X Windows</td>
<td>unix / C, networked GUI, vendor independent.</td>
</tr>
<tr>
<td>1985</td>
<td>MS Windows</td>
<td>intel GUI, tiled windows ... metro UI</td>
</tr>
<tr>
<td>1989</td>
<td>Next</td>
<td>OOP GUI interface builder</td>
</tr>
<tr>
<td>1992</td>
<td>3D GUIs</td>
<td>Xerox Parc: data wall, cone tree ...</td>
</tr>
<tr>
<td>1993</td>
<td>Newton</td>
<td>handheld GUI (tablet)</td>
</tr>
<tr>
<td>1994</td>
<td>Mosaic</td>
<td>Graphical Web browser (now Firefox)</td>
</tr>
<tr>
<td>1995</td>
<td>Java AWT</td>
<td>System independent local</td>
</tr>
<tr>
<td>1996</td>
<td>Nokia communicator</td>
<td>smart phone</td>
</tr>
<tr>
<td>2003</td>
<td>.NET</td>
<td>Language independent GUI ... WPF</td>
</tr>
<tr>
<td>2010</td>
<td>Apple iPad</td>
<td>Gesture i/o, mobile computing</td>
</tr>
<tr>
<td>now</td>
<td>wearables</td>
<td>ubiquitous, location based computing</td>
</tr>
<tr>
<td>soon</td>
<td>???</td>
<td>(Augmented Reality), smart bikes</td>
</tr>
<tr>
<td></td>
<td>soon</td>
<td>human embedded computing, iCyborg</td>
</tr>
</tbody>
</table>
Application makes display requests to system
System display is an event to the user
User responds to display with input requests (mouse, keyboard...)
System routes user input to application as events
Application handles events and may make display requests...
MVC architecture

SmallTalk’s window concept had 3 object components

**Model:** object to be viewed or modified

**View:** object that determines how the model is to be displayed “draw to display requests”

**Controller:** object that handles user interaction with "window” “handle events”
Consider a binary tree visual simulation:

1. model internal binary tree algorithms
2. views graphic indented text
3. controllers keyboard voice input

$ add 30
$ delete 15
$ "add 30"
"delete 15"
Model View architecture

MVC variants used in modern GUIs

Java UI-delegate combines MVC view-controller aspects

UI objects know how to: draw themselves, handle events, determine their size ... 

Java UI classes can have associated model objects

MS .NET uses "controls", "views", and models. Views can be complex controls.

I will use model and view as references to the model and the view-controller concepts.
HelloSwing example

create and show window

wait for user interaction

Click application button

redisplay window

Click system menu close button

Application's lifetime

close and destroy window
HelloSwing :: enter main()
HelloSwing :: enter HelloSwing()
HelloSwing :: leave HelloSwing()
HelloSwing :: leave main()
HelloSwing :: actionPerformed()
HelloSwing :: actionPerformed()

/**
 * HelloSwing Java example program for CS 585.
 * Frame window with a button.
 * Click button and label changes.
 * Trace statements are added to illustrate order of method calls.
 *
 * Mike Barnes 8/18/2014
 */

// need to import awt components and events and swing components.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
Look at HelloSwing.html generated with javadoc: inheritance, documentation.

```java
/**
 * HelloSwing is an application - subclass Frame for window
 */
public class HelloSwing extends JFrame {
    private JButton aButton;
    private boolean toggle = true;

    //All applications must have a main method.
    public static void main(String args[]) {
        System.out.println("HelloSwing :: enter main()");
        HelloSwing app = new HelloSwing("Hello Java");
        app.setSize(300,100);
        app.setVisible(true);
        // need to be able to close the window.
        app.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        System.out.println("HelloSwing :: leave main()");
    }
```

// the constructor of the frame window
public HelloSwing(String frameTitle) {
    super(frameTitle);
    System.out.println("HelloSwing :: enter HelloSwing()");
    Container contentPane = getContentPane();
    JButton aButton = new JButton("Hello CS 585 !");
    // listen for JButton events
    aButton.addActionListener(new JButtonListener());
    contentPane.add(aButton, "Center");
    System.out.println("HelloSwing :: leave HelloSwing()");
}

// the "callback function" to handle button press events
class JButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        System.out.println("HelloSwing :: actionPerformed()");
        if (toggle)
            aButton.setText("Press me again");
        else
            aButton.setText("Hello CS 585 !");
        toggle = !toggle;
    }
}

}