Graphical user interfaces

- models
  - X Window system
  - Java Swing
  - Visual Basic
  - Tcl/Tk (maybe)

- ideas
  - interface components: widgets, controls, objects, ...
  - methods, properties
  - events: loops and callbacks
  - geometry and layout management
  - use of hierarchy, inheritance

- the GUI is the biggest chunk of code in many applications
  - libraries try to make it easier
  - development environments and wizards and builders try to make it easier
  - it’s still hard

X Windows (Bob Scheifler, Jim Gettys, mid-1980’s)

- client-server over a network
  - works on single machine too, with IPC

- variants:
  - X terminal (e.g., SunRay): server is standalone
  - workstation: server is on same processor as clients
  - remote clients, local server
  - Exceed: server on PC, clients on (usually) Unix

- window manager is just another client, but with more properties
  - clients have to let the window manager manage
  - InterClient Communications Conventions Manual
X Windows model (www.x.org)

- **server** runs on the local machine
  - accepts network or local client request and acts on them
  - creates, maps, and destroys windows
  - writes and draws in windows
  - manages keyboard, mouse, and display
  - sends keyboard and mouse events back to proper clients
  - replies to information requests
  - reports errors

- **client** application
  - written with X libraries (i.e., Xlib, Xt)
  - uses the X protocol to
    - send requests to the server
    - receive replies, events, errors from server

- **protocol, messages**
  - requests: clients make requests to the server
    - e.g., Create Window, Draw, Iconify, ...
  - replies: server answers queries ("how big is this?"
  - events: server forwards events to client
    - typically keyboard or mouse input
  - errors: server reports request errors to client

X Windows programming model

- Xlib provides client-server communication
- intrinsics provide basic operations for building and combining widgets
- widgets implement user interface components
  - buttons, labels, dialog boxes, menus, ...
  - multiple widget sets, e.g., Motif
- application uses all of these layers
**Xlib: bottom level library**

- **basic mechanisms for**
  - requests from client to server: "draw on window", "how big is this?"
  - replies from server to client: "this big"
  - events from server to client: "button 1 pushed", "window exposed"
  - error reports: "out of memory"

- **basic Xlib-level client program**
  - connect client to server: `XOpenDisplay()`
  - get info about screen, compute desired size for window;
    hints, not mandatory; the WM is in charge
  - create window: `XCreateSimpleWindow()`
  - set standard properties for window manager
    sizes, window name, icon name, ...
  - select events to be received and discarded
  - create "graphics context" for storing info on color, depth, ...
    things that don’t change from request to request
    server caches this info to cut down traffic
  - display window: `XMapWindow()`
    causes it to appear; up to this point it hasn’t
  - loop on events

**Events**

- client registers for events it cares about
- events occur asynchronously
- queued for each client
- client has to be ready to handle events any time
  - mouse buttons or motion
  - keyboard input
  - window moved or reshaped or exposed
  - 30-40 others
- information comes back to client in a giant union
  `XEvent`
  ```c
  XEvent myevent;
  for (;;) {
    XNextEvent(mydisplay, &myevent);
    switch (myevent.type) {
      case ButtonPress: ...
      ...
  }
  ```
Hello world in X toolkit/ Motif widgets

```c
#include <Xm/XmAll.h>

void main(int argc, char *argv[])
{
    Widget toplevel, main_w, button;
    XtAppContext app;
    XtSetLanguageProc(NULL, NULL, NULL);
    toplevel = XmVaAppInitialize(&app, "main", NULL, 0,
        &argc, argv, NULL, NULL);
    main_w = XmVaCreateManagedWidget("main_w",
        xmMainWindowWidgetClass,
        toplevel, XmNscrollingPolicy,
        XmAUTOMATIC, NULL);
    button = XmVaCreateWidget("Hello World",
        xmLabelWidgetClass, main_w, NULL);
    XtManageChild(button);
    XtRealizeWidget(toplevel);
    XtAppMainLoop(app);
}
/* cc x.c -lxm -lxt -lx11 */
```

Hello world in GTK (plus ça change...)

```c
#include <gtk/gtk.h>

static void hello( GtkWidget *widget, gpointer data )
{
    g_print("Hello World\n");
}
static gboolean delete_event( GtkWidget *widget, GdkEvent  *event,
    gpointer  data )
{
    g_print("delete event occurred\n");
    return TRUE;
}
static void destroy( GtkWidget *widget, gpointer data )
{
    gtk_main_quit ();
}
int main(int argc, char *argv[])
{
    GtkWidget *window;
    GtkWidget *button;
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    g_signal_connect (G_OBJECT (window), "delete_event",
        G_CALLBACK (delete_event), NULL);
    g_signal_connect (G_OBJECT (window), "destroy",
        G_CALLBACK (destroy), NULL);
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);
    button = gtk_button_new_with_label ("Hello World");
    g_signal_connect (G_OBJECT (button), "clicked",
        G_CALLBACK (hello), NULL);
    g_signal_connect_swapped (G_OBJECT (button), "clicked",
        G_CALLBACK (gtk_widget_destroy),
        G_OBJECT (window));
    gtk_container_add (GTK_CONTAINER (window), button);
    gtk_widget_show (button);
    gtk_widget_show (window);
    gtk_main ();
    return 0;
}
```
Graphical user interfaces in Java

- interfaces built from components
  - buttons, labels, text areas, lists, menus, dialogs, ...
  - canvas: graphics for drawing and image rendering
- each component has
  - properties: size, position, visibility, text, font, color, ...
  - methods: things it will do, e.g., change properties
  - events: external stimuli it responds to
- containers: hold components and containers
- layout managers: control size, placement of objects within a container

(Campione & Walrath Java Tutorial)

Which GUI package?

- Java has two major GUI packages
- Abstract Window Toolkit (AWT)
  - the original Java 1.0 version
  - very slow and weak, but simple
  - exists everywhere, though deprecated (correctly)
  - Java 1.1 has a newer event model
    - slightly more complicated, somewhat faster
- Swing
  - for Java 1.2 and beyond
  - bigger, more complicated: many more components
  - "pluggable" look and feel
  - more powerful, faster
  - not as widespread, not in most browsers
    - but available as a plug-in
- we’ll mostly use Swing
Component object hierarchy

Object
  -> Component
    -> Container
      -> JComponent
      -> Jpanel
      -> JLabel
      -> JButton
      -> JTextComponent
        -> JTextField
        -> JPasswordField
        -> JTextArea
        -> ...

- containers hold components and containers
  - used to build up nested structures
  - JFrame top-level window
  - JPanel general container for components & containers
    put JPanels in a JFrame
  - JMenuBar for menu bar across top of frame

- individual components like JButton, JTextField...
  - respond to events
  - have methods for other behaviors
  - have get and set methods for accessing properties
    like size, color, font
Layout hierarchy

- JFrame holds one or more JPanels
- JPanel holds components and other JPanels
- JPanel used for layout
  - add() method adds components
  - uses a LayoutManager that lays out components
  - layout manager can be set to one of several

Events

- stuff happens
  - mouse motion, button push, button release, ...
  - scrollbar fiddled
  - keyboard keypress, release, shift key, etc.
  - component got or lost focus
  - window iconified, uniconified, hidden, exposed, moved, reshaped, killed
  - etc.

- each such event is passed to event-handling mechanism in the program

- program can decide what to do with it
Events (1.1 model)

- components register to receive (listen for) events that they are interested in
- AWT runs a thread that watches for events like button push, mouse motion or click, keyboard, ...
- each event is passed to the listener that registered for it

- `obj.addActionListener(this)`
  - tells `obj` to notify this container when event happens
  - usually called by container that contains object that will get the event

- `void actionPerformed(ActionEvent e) { ... }`
  - called when event occurs
  - determine type or instance that caused event
  - handles it

- different kinds of listeners for different sources
  - keyboard, mouse, mouse motion, window, ...

Example 1: Buttons and labels

- after it starts:

  ![Image of a window with buttons: Count, New Count, Quit, and a counter]

- after Count button is pushed 3 times:

  ![Image of a window with buttons: Count has a value 3, New Count, Quit]

- after Quit button is pushed:
Example 1 events, layout

```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class ex1 extends JFrame
    implements ActionListener {
    int count;
    JLabel lab;
    JButton bcount, bquit;
    public static void main(String[] args) {
        ex1 a = new ex1();
    }
    ex1() {
        setTitle("ex1");
        JPanel p1 = new JPanel();
        lab = new JLabel("Counter");
        p1.add(lab);
        bcount = new JButton("Count",
            new ImageIcon("new.gif"));
        bcount.addActionListener(this);
        bquit = new JButton("Quit");
        bquit.addActionListener(this);
        JPanel p2 = new JPanel();
        p2.add(bcount);  p2.add(bquit);
        getContentPane().setLayout(new BorderLayout());
        getContentPane().add(p1, BorderLayout.NORTH);
        getContentPane().add(p2, BorderLayout.SOUTH);
        pack();
        show();
    }
    public void actionPerformed(ActionEvent ae) {
        System.out.println(ae.getActionCommand());
        if (ae.getActionCommand().equals("Count")) {
            count++;
            lab.setText(Integer.toString(count));
        } else if (ae.getSource() == bquit) {
            System.exit(0);
        }
    }
}
```

Example 1, continued

- lots of steps to set up a control:
  - declare an object, like Button
  - create it with new
  - add it to a container
  - add an ActionListener
  - test for the action in actionPerformed
Layout managers

- control size, position, padding, stretching & shrinking, etc., for containers
- each container has a default layout manager
  - change with setLayout method
    - or at creation
      JPanel jp = new JPanel(new BorderLayout());

  - FlowLayout
    - fills area left to right in rows
    - each row can be centered, left or right adjusted
  - BorderLayout
    - fills North, South, East, West, and Center
  - GridLayout
    - regular array of rows and columns
    - specify number of each

  - CardLayout
    - multiple windows that all occupy the same space
    - like tabs in Microsoft interfaces
  - GridBagLayout
    - complicated grid layout
    - weighting factors on component areas, stretching, etc.

Flow Layout

- default for Panels

```java
public class layout1 extends JFrame {

  public static void main(String[] args) {
    layout1 a = new layout1();
    a.setTitle("layout1: flow");
    JPanel p = new JPanel();
    JButton b1 = new JButton("One");
    JButton b2 = new JButton("Two");
    JButton b3 = new JButton("Three");
    JButton b4 = new JButton("Four");
    JButton b5 = new JButton("Five");
    p.add(b1);
    p.add(b2);
    p.add(b3);
    p.add(b4);
    p.add(b5);
    a.getContentPane().add(p);
    a.pack();
    a.show();
  }
}
```
Border Layout

```java
public class layout2 extends JFrame {
    public static void main(String[] args) {
        layout2 a = new layout2();
        a.setTitle("layout2: border");
        JPanel p = new JPanel();
        p.setLayout(new BorderLayout());
        p.add(new JButton("north button"), BorderLayout.NORTH);
        p.add(new JButton("south button "), BorderLayout.SOUTH);
        p.add(new JButton("east"), BorderLayout.EAST);
        p.add(new JButton("westernmost button"), BorderLayout.WEST);
        p.add(new JButton("center button"), BorderLayout.CENTER);
        a.getContentPane().add(p);
        a.pack();
        a.show();
    }
}
```

Grid Layout

```java
public class layout3 extends JFrame {
    public static void main(String[] args) {
        layout3 a = new layout3();
        a.setTitle("layout3: grid");
        JPanel p = new JPanel();
        p.setLayout(new GridLayout(3, 2));
        p.add(new Button("One"));
        p.add(new Button("Two "));
        p.add(new Button("Three"));
        p.add(new Button("Four"));
        p.add(new Button("Five"));
        a.getContentPane().add(p);
        a.pack();
        a.show();
    }
}
```