

User interface design

"The interface is the product."

Jef Raskin

originator of the Apple Macintosh

"A user interface is well designed when the program behaves exactly how the user thought it would."

Joel Spolsky

User Interface Design for Programmers

Uniformity, consistency

- **uniform visual appearance throughout: sizes, colors, fonts, ...**
 - *CSS* is your friend for web pages
- **always put similar items in standard places**
 - File / Edit / View ... / Help
 - OK / Cancel at the bottom right
 - consistency with other systems if it makes sense
 - use the local look and feel
- **follow conventions**
 - ... means "more to come"
 - grayed out means inactive
 - checkmark means currently selected
 - ...

Legibility

- not everyone is 18 years old!
- use large enough text sizes
- use legible fonts
- use contrasting colors
 - don't use dark blue letters on black background!
- don't assume a screen size
 - text should adapt to screen size, not require scrolling
- mobile is different but the same principles apply

Ease of use

- **pick good defaults**
- **don't get too deep**
 - multi-level menus
 - drop-downs that don't fit on the screen
- **remember useful information**
 - previous text entries
 - position in file system
 - settings
- **but do it consistently**
- **provide text-based alternatives**
 - let me type filenames instead of forcing a dialog
- **think about accessibility**
- **think about other languages and cultures**

Safety first

- **don't do irrevocable actions without confirmation**
 - don't quit without warning if changes are not saved
- **but do it right**
 - don't ask about saving if there were no changes
Excel and Word both do this wrong
 - show an indicator of whether changes have occurred
- **provide a way to interrupt long-running computations safely**
- **watch out for security warnings that are clicked automatically**

Usability testing *(Krug, Rocket Surgery Made Easy)*

- figure out a small set of important tasks
- make scenarios that tell the user what to perform
 - but not how to do it
- try it yourself
- write it down so the user can refer to it
- get the user to talk out loud about his/her thought processes while performing it

Useful reading

- **Windows user experience interaction guidelines:**
 - <http://msdn.microsoft.com/en-us/library/windows/desktop/aa511258.aspx>
- <http://www.nngroup.com/articles/>
 - (Don Norman, Jakob Nielsen, Bruce Tognazzini, ...)
- goodui.org
- **Steve Krug, *Rocket Surgery Made Easy***
- **Joel Spolsky, *User Interface Design for Programmers***
- **World's worst web site (two of a million similar sites)**
 - <http://www.angelfire.com/super/badwebs/>
 - <http://www.webpagesthatsuck.com/>

Graphical user interfaces: what the user sees and uses

- **fundamental ideas**

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

- **examples of GUI-building systems**

- X Window system; GTK
- Java Swing; GWT, Android
- Tcl/Tk, with bindings for Python, Ruby, Perl, ...

- XCode for MacOSX and iThings
- Android Studio (based on IntelliJ)
- Eclipse for Java and many other languages
- Microsoft Visual Studio for C++, C#, VB, ..

- HTML, CSS, Javascript (jQuery, Bootstrap, React, Angular, and a thousand others)

Graphical user interfaces

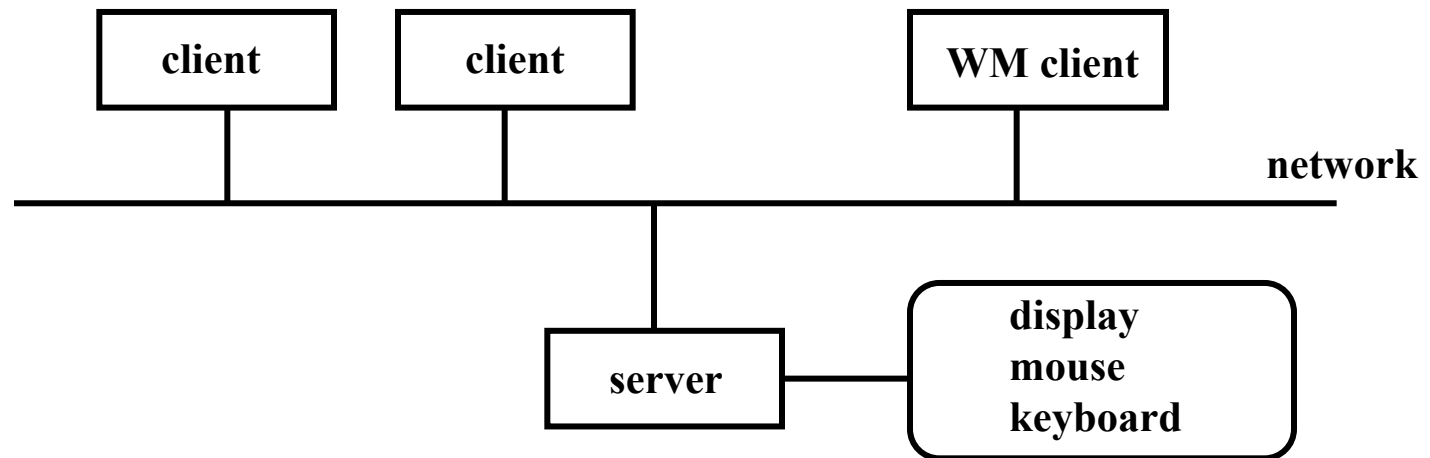
- **interfaces are built from components/widgets**
 - buttons, labels, text areas, lists, menus, dialogs, ...
 - canvas: graphics for drawing and image rendering
- **each component has**
 - properties: size, position, visibility, text, font, color, ...
 - methods: things it will do or that you can do to it, 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**
 - some programmable, some purely by drawing
 - how they adapt to changes like reshaping
- **most GUI systems have the same basic ideas and building blocks, but with many differences in details**

Methods, properties, events (Javascript)


```
<head>
<script>
function setfocus() { document.srch.q.focus(); }
</script>
</head>
<BODY onload='setfocus();'>
<H1>Basic events on forms</H1>
<form action="http://www.google.com/search" name=srch>
<input type=text size=25 name=q id=q value=""
    onmouseover='setfocus()'>
<input type=button value="Google" name=but
    onclick='window.location=
        "http://www.google.com/search?q="+srch.q.value'>
<input type=button value="Wikipedia" name=but
    onclick='window.location=
        "http://en.wikipedia.com/wiki/"+srch.q.value'>
<input type=reset onclick='srch.q.value=""; >
</form>
```

X Windows (Bob Scheifler & Jim Gettys, 1984)

- **client-server over a network**
 - works on single machine too, with inter-process communication

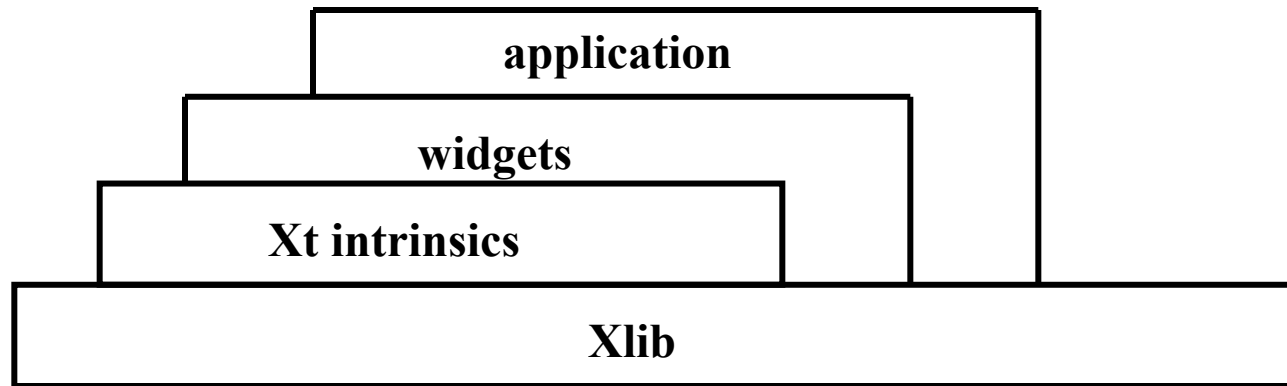


- **variants:**
 - "X terminal": server is only process on server, clients are all remote
 - workstation: server is on same processor as clients
 - "PC X server": server on Windows, clients on (usually) Unix
- **window manager is just another client, but with more properties**
 - clients have to let the window manager manage
 - permits multiple workspaces / virtual windows / virtual desktops

X Windows model (www.x.org)

- **server runs on the local machine**
 - accepts network (or local) client requests 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, GTK, ...)
 - uses the X protocol to send requests to the server, and 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**
 - initial connection of client to server, window creation, window properties, event mask, ...
 - sends client requests to server: draw, get size, ...
 - sends server responses, errors, etc., to client
 - send events from server, like button push, key press, window expose, ...
- **Xt intrinsics provide basic operations for creating and combining widgets**
- **widgets implement user interface components**
 - buttons, labels, dialog boxes, menus, ...
 - widget set is a group of related widgets with common look and feel, e.g., Motif, GTK
- **applications and libraries can use all of these layers**

Events

- client registers with windows system 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 called XEvent, placed in a queue
- "event loop" processes the queue

```
Xevent myevent;  
for (;;) {  
    XNextEvent(mydisplay, &myevent);  
    switch (myevent.type) {  
    case ButtonPress: ...  
    ...  
    }
```

Hello world in X toolkit / Xlib

```
#include<X11/Xlib.h>
#include<stdio.h>
#include<stdlib.h>
int main() {
    Display *dpy;
    Window rootwin;
    Window win;
    Colormap cmap;
    XEvent e;
    int scr;
    GC gc;
    if(!(dpy=XOpenDisplay(NULL))) {
        fprintf(stderr, "ERROR: could not open display\n");
        exit(1);
    }
    scr = DefaultScreen(dpy);
    rootwin = RootWindow(dpy, scr);
    cmap = DefaultColormap(dpy, scr);
    win=XCreateSimpleWindow(dpy, rootwin, 1, 1, 100, 50, 0,
        BlackPixel(dpy, scr), BlackPixel(dpy, scr));
    XStoreName(dpy, win, "hello");
    gc=XCreateGC(dpy, win, 0, NULL);
    XSetForeground(dpy, gc, WhitePixel(dpy, scr));
    XSelectInput(dpy, win, ExposureMask|ButtonPressMask);
    XMapWindow(dpy, win);
    while(1) {
        XNextEvent(dpy, &e);
        if(e.type==Expose && e.xexpose.count<1)
            XDrawString(dpy, win, gc, 10, 10, "Hello World!", 12);
        else if(e.type==ButtonPress) break;
    }
    XCloseDisplay(dpy);
    return 0;
}
```

Hello world in Java Swing

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

public class helloworld extends JFrame {

    public static void main(String[] args) {
        helloworld a = new helloworld();
    }

    helloworld() {
        JButton b = new JButton("hello world");
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                System.exit(0);
            }
        });
        getContentPane().add(b);
        pack();
        setVisible(true);
    }
}
```


Component object hierarchy

Object

Component

Container

JComponent

JPanel

JLabel

JButton

JTextComponent

 JTextField

 JFormattedTextField

 JPasswordField

 JTextArea

 JEditorPane

 JTextPane

- **containers hold components & containers, used to build up nested structures**
 - JFrame: top-level window
 - JPanel: general container for components & containers
 - JMenuBar for menubar across top of JFrame
 - JToolBar for toolbar, possibly floating
- **individual components like JButton, JTextArea, ...**
 - 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 to the panel
 - panel uses a LayoutManager that lays out components
 - layout manager can be set to one of several choices

The screenshot shows a Java Swing window with a title bar and three window control buttons. The main content area is enclosed in a red dashed box, which is labeled 'JPanel' with an arrow pointing to the top-right corner. Inside this dashed box, there are three text input fields: 'Principal' with the value '200000', 'Interest Rate' with the value '6', and 'Monthly Payment' with the value '2000'. Below these fields is a table titled 'Payment Schedule:' with four columns. The table contains 13 rows of data, with the final row showing a total of 77951.56 and 200000.00. To the right of the table, there are three buttons: 'Update', 'Clear', and 'Quit'. A red dashed box labeled 'JPanel' with an arrow points to the bottom-left corner of the window. Another red dashed box labeled 'JPanel' with an arrow points to the bottom-right corner of the window.

	Principal	Interest Rate	Monthly Payment
127	125.33	1874.67	23192.24
128	115.96	1884.04	21308.20
129	106.54	1893.46	19414.74
130	97.07	1902.93	17511.82
131	87.56	1912.44	15599.37
132	78.00	1922.00	13677.37
133	68.39	1931.61	11745.76
134	58.73	1941.27	9804.49
135	49.02	1950.98	7853.51
136	39.27	1960.73	5892.78
137	29.46	1970.54	3922.24
138	19.61	1980.39	1941.85
139	9.71	1941.85	0.00
		77951.56	200000.00

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 the event-handling mechanism in the program**
- **the program can decide what to do with it**

Events in Swing

- **components register to receive (listen for) events that they are interested in:**

```
    JButton jb = new JButton("whatever");
```

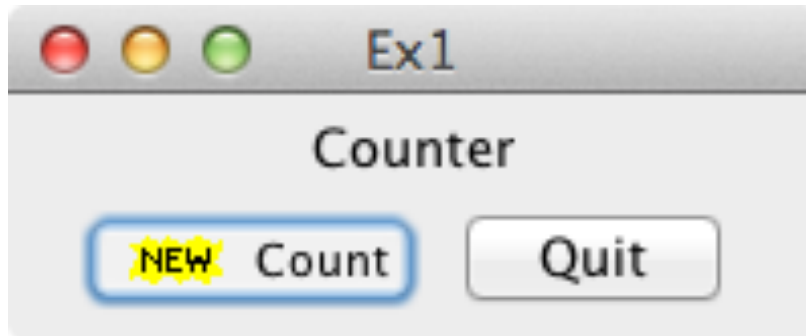
```
    jb.addActionListener(this);
```

- tells `jb` to notify `this` container when event happens
i.e., sets up a callback
- usually called by container that contains object that will get the event
- **a thread watches for events like button push, mouse motion or click, key down or up, ...**
- **when event occurs, listener's `actionPerformed` is called**
 - from component where event occurs (e.g., button instance) when it does
- **handler determines type or instance that caused event, does appropriate action**

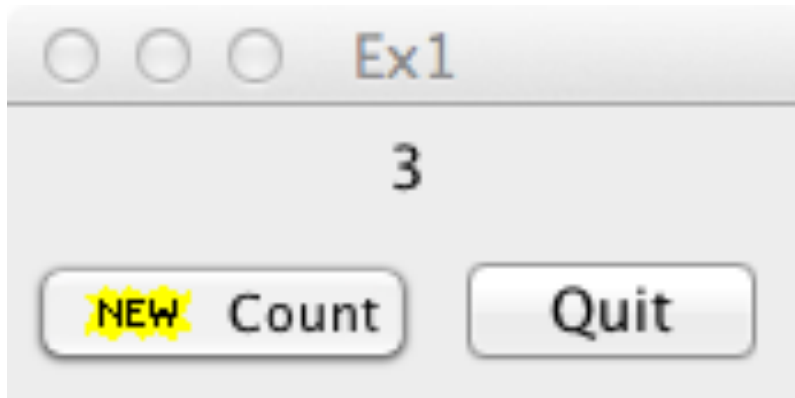
```
    actionPerformed(ActionEvent e) { ... }
```
- **different kinds of listeners for different sources**
 - keyboard, mouse, mouse motion, window, ...

Example 1: Buttons and labels

- after it starts:



- after Count button is pushed 3 times:



- after Quit button is pushed:

Example 1 events, layout

```
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");
    lab = new JLabel("Counter");
    JPanel p1 = new JPanel(); 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();
    setVisible(true);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
}
```

Example 1, continued

```
// the one function of the ActionListener interface:
public void actionPerformed(ActionEvent ae) {
    System.out.println(ae.getActionCommand());
    if (ae.getActionCommand().equals("Count")) { // by content
        count++;
        lab.setText(Integer.toString(count));
    } else if (ae.getSource() == bquit) { // by object name
        System.exit(0);
    }
}
```

- **five steps to set up a GUI component:**
 - declare an object, like Button
 - create it with new
 - add it to a container
 - add an ActionListener to catch events
 - handle events in actionPerformed
- **information is spread all over the place**

Anonymous inner classes

- an unnamed class defined inside another class

```
bcount = new JButton("Count", new ImageIcon("new.gif"));
bcount.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
        count++;
        lab.setText(Integer.toString(count));
    }
});
```

- equivalent to this, without separate declaration and name

```
class foo implements ActionListener {
    public void actionPerformed(ActionEvent ae) {
        ...
    }
}
button.addActionListener(new foo());
```


Geometry / Layout manager approaches

- **Java Swing**
 - position by imperative code, with 8 standard layout managers
 - graphical layout by NetBeans IDE
- **Tk (from Tcl, Python, etc.)**
 - mostly declarative: position relative to other positioned objects
- **VB (pre .NET)**
 - mostly draw on a screen: absolute positioning
- **C#, current VB**
 - drawing objects creates imperative code as side effect
 - can use either method to do layout
- **iPhone**
 - Interface Builder mostly drawing on screen
 - can create objects, position them, etc., by imperative commands
- **Android**
 - declarative positioning specified in XML
 - can create objects, position them, etc., by imperative commands
- **HTML**
 - mostly implicit via box model with <div> tags

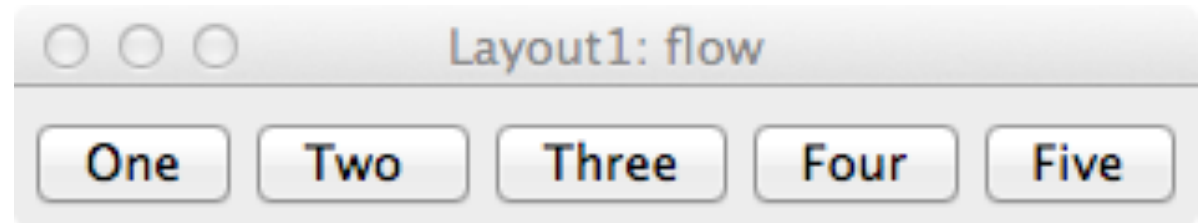
Layout managers in Swing

- control container size, position, padding, stretch & shrink, etc.,
- each container has a default layout manager
 - set it at creation or change it later with `setLayout` method

```
 JPanel jp = new JPanel(new BorderLayout());  
 jp.setLayout(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
(`BorderLayout.NORTH`, etc.)
- **GridLayout**
 - regular array of specified number of rows and columns
- **CardLayout**
 - multiple windows that all occupy the same space
 - usually selected with tabs or combo boxes
- etc., etc.

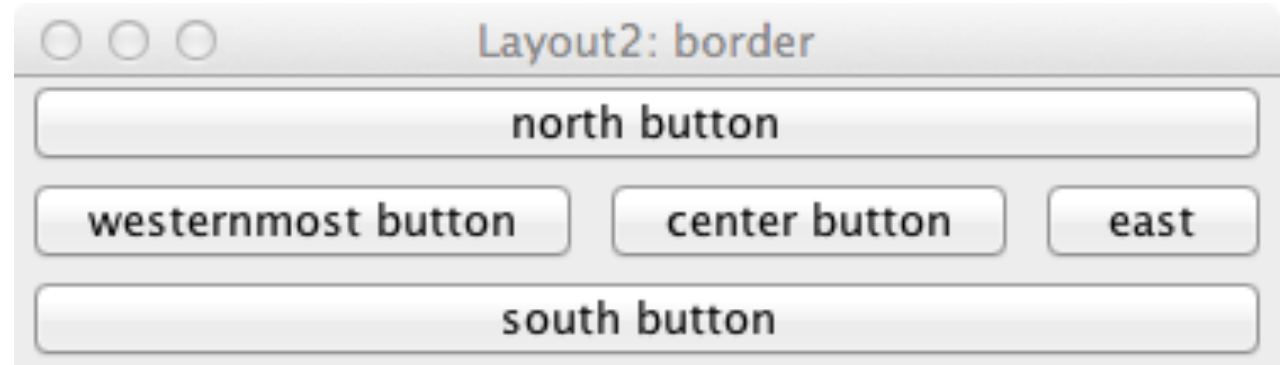
Flow Layout

- default for Panels



```
public class Layout1 extends JFrame {
    public static void main(String[] args) {
        Layout1 a = new Layout1();
        a.setTitle("Layout1: flow");
        JPanel p = new JPanel();
        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.setVisible(true);
    }
}
```

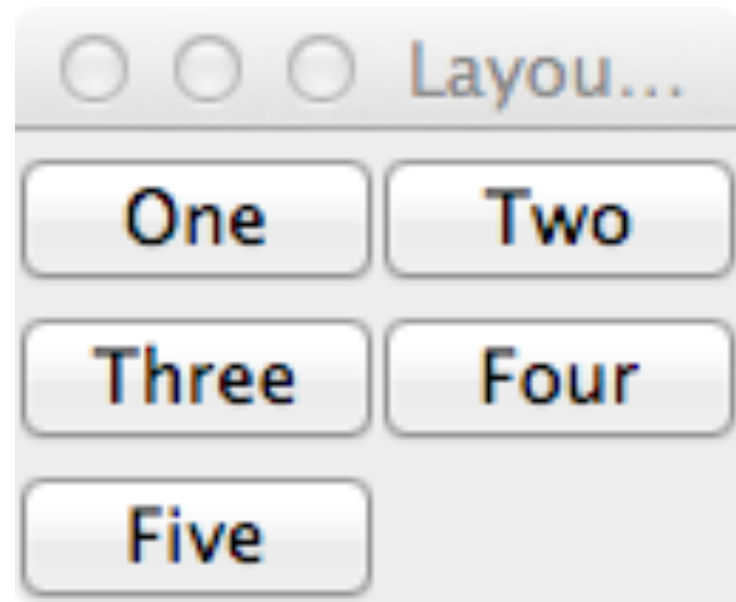
Border Layout



```
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.setVisible(true);
    }
}
```

Grid Layout

```
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.setVisible(true);  
    }  
}
```



Layout hierarchy

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

The screenshot shows a Java Swing window with a title bar and three window control buttons. The main content area is enclosed in a red dashed box, which is labeled "JPanel" with an arrow pointing to the top-right corner. Inside this dashed box, there are three text input fields: "Principal" with the value "200000", "Interest Rate" with the value "6", and "Monthly Payment" with the value "2000". Below these fields is a table titled "Payment Schedule:" with four columns of numerical data. The table contains 13 rows of data, with the final row showing a total of 77951.56 and 200000.00. To the right of the table, there are three buttons: "Update", "Clear", and "Quit". A red dashed box labeled "JPanel" is positioned at the bottom-left corner of the window, with an arrow pointing to the table area. Another red dashed box labeled "JPanel" is positioned at the bottom-right corner of the window, with an arrow pointing to the buttons area.

	Principal	Interest Rate	Monthly Payment
127	125.33	1874.67	23192.24
128	115.96	1884.04	21308.20
129	106.54	1893.46	19414.74
130	97.07	1902.93	17511.82
131	87.56	1912.44	15599.37
132	78.00	1922.00	13677.37
133	68.39	1931.61	11745.76
134	58.73	1941.27	9804.49
135	49.02	1950.98	7853.51
136	39.27	1960.73	5892.78
137	29.46	1970.54	3922.24
138	19.61	1980.39	1941.85
139	9.71	1941.85	0.00
		77951.56	200000.00

Layout hierarchy

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

The screenshot shows a Java Swing window with three input fields at the top: "Principal" (200000), "Interest Rate" (6), and "Monthly Payment" (2000). Below these is a table titled "Payment Schedule:" with 19 rows of data. To the right of the table are three buttons: "Update", "Clear", and "Quit".

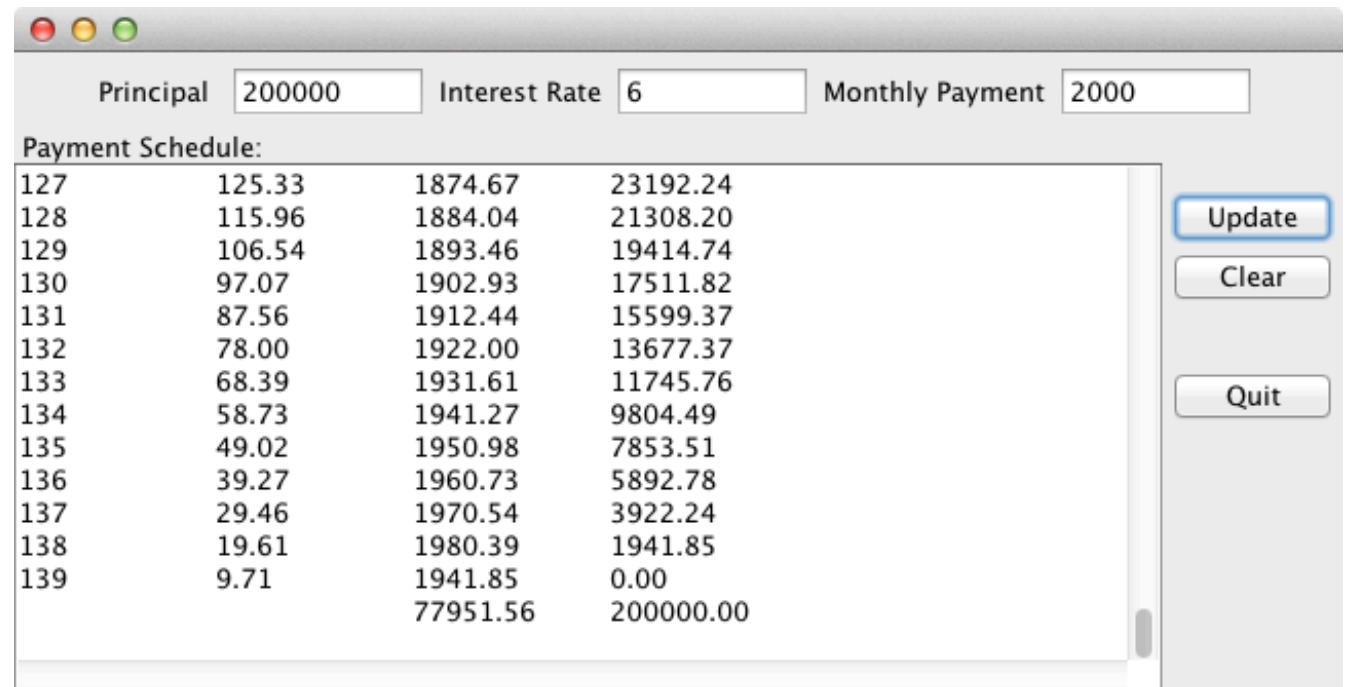
Annotations in the image:

- Flow layout**: Points to the input fields.
- Grid Layout**: Points to the table.
- Border Layout**: Points to the buttons.

	Principal	Interest Rate	Monthly Payment
125	125.33	1874.67	23192.24
127	115.96	1884.04	21308.20
128	106.54	1893.46	19414.74
129	97.07	1902.93	17511.82
130	87.56	1912.44	15599.37
131	78.00	1922.00	13677.37
132	68.39	1931.61	11745.76
133	58.73	1941.27	9804.49
134	49.02	1950.98	7853.51
135	39.27	1960.73	5892.78
136	29.46	1970.54	3922.24
137	19.61	1980.39	1941.85
138	9.71	1941.85	0.00
139		77951.56	200000.00

Example 2: Text components

- **JTextField**
 - single line for input
 - main interesting event is pushing Return
- **JTextArea**
 - multiple lines; can add scrolling
 - can edit in place
 - can change size and font for whole area but not parts
 - fancier JTextComponents for editing, display of different sizes and fonts, HTML, etc.



Principal Interest Rate Monthly Payment

Payment Schedule:

127	125.33	1874.67	23192.24
128	115.96	1884.04	21308.20
129	106.54	1893.46	19414.74
130	97.07	1902.93	17511.82
131	87.56	1912.44	15599.37
132	78.00	1922.00	13677.37
133	68.39	1931.61	11745.76
134	58.73	1941.27	9804.49
135	49.02	1950.98	7853.51
136	39.27	1960.73	5892.78
137	29.46	1970.54	3922.24
138	19.61	1980.39	1941.85
139	9.71	1941.85	0.00
		77951.56	200000.00

Example 2 code excerpts

```
class Mtg extends JFrame implements ActionListener {
    JLabel lprin = new JLabel("Principal ");
    JTextField tprin = new JTextField(7);
    JLabel lrate = new JLabel("Interest Rate");
    JTextField trate = new JTextField(7);
    JLabel lmpay = new JLabel("Monthly Payment");
    JTextField tmpay = new JTextField(7);

    JLabel lsched = new JLabel("Payment Schedule:");
    JTextArea tpay = new JTextArea(15, 45);

    JButton update = new JButton("Update");
    JButton clear = new JButton("Clear");
    JButton quit = new JButton("Quit");

    public static void main(String[] args) {
        Mtg m = new Mtg();
    }
}
```

Example 2, page 2

```
Mtg() {
    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
    // top row of entry boxes
    JPanel ptop = new JPanel();
    ptop.add(lprin); ptop.add(tprin); ptop.add(lrate);
    ptop.add(trate); ptop.add(lmpay); ptop.add(tmpay);
    tprin.setToolTipText("Enter principal amount");
    trate.setToolTipText("Enter yearly interest rate ...");
    tmpay.setToolTipText("Enter monthly payment");
    // text area for payment schedule
    JScrollPane jsp = new JScrollPane(tpay,
        JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
        JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
    JPanel pctr = new JPanel(new BorderLayout());
    pctr.add(lsched, BorderLayout.NORTH);
    pctr.add(jsp, BorderLayout.CENTER);
}
```

Example 2, page 3

```
// buttons on right
JPanel pr = new JPanel(new GridLayout(0,1));
pr.add(new JLabel()); // spacer
pr.add(update); pr.add(clear);
pr.add(new JLabel()); // spacer
pr.add(quit);
JPanel pright = new JPanel(new BorderLayout());
pright.add(pr, BorderLayout.NORTH); // pack at top
update.addActionListener(this);
clear.addActionListener(this);
quit.addActionListener(this);
update.setToolTipText("Update payment schedule");
clear.setToolTipText("Clear payment schedule");
// overall layout
Container cp = getContentPane();
cp.add(ptop, BorderLayout.NORTH);
cp.add(pctr, BorderLayout.CENTER);
cp.add(pright, BorderLayout.EAST);
pack();
setVisible(true);
}
```

Example 2, page 4

```
public void actionPerformed(ActionEvent e) {  
    if (e.getSource() == quit) {  
        System.exit(0);  
    } else if (e.getSource() == update) {  
        tpay.setText(pay());  
    } else if (e.getSource() == clear) {  
        tpay.setText("");  
    }  
}
```

Example 2, page 5

```
String pay() {
    double mp = Double.parseDouble(tmpay.getText());
    double prin = Double.parseDouble(tprin.getText());
    double mrate = Double.parseDouble(trate.getText())/12/100;
    double totint = 0, totprin = 0;
    String s = "";
    for (int i = 1; i <= 500; i++) {
        double Int = prin * mrate;
        double dp = mp - Int;    // decrease of principal
        if (prin - dp > 0) {
            prin -= dp;
        } else {
            dp = prin;
            prin = 0;
        }
        s += String.format("%d\t%.2f\t%.2f\t%.2f\n",i, Int, dp, prin);
        totint += Int;
        totprin += dp;
        if (prin <= 0)
            break;
    }
    s += String.format("\t\t%.2f\t%.2f\n", totint, totprin);
    return s;
}
```



Package Explorer

- ApiDemos
- Axcel
- Hello2
 - src
 - com.example.hello
 - Hello2.java
 - R.java
 - Android Library
 - assets
 - res
 - AndroidManifest.xml
- Home
- NotesList
- SkeletonActivity

```

package com.example.hello;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class Hello2 extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        TextView tv = new TextView(this);
        tv.setText("Hello, world");
        setContentView(tv);
    }
}
    
```

Hello world in Android

Problems | Javadoc | Declaration | Console | Outline | Properties

```

Android
[2009-03-21 11:28:32 - Hello2] adb is running normally.
[2009-03-21 11:28:32 - Hello2] Launching: com.example.hello.Hello2
[2009-03-21 11:28:32 - Hello2] Automatic Target Mode: launching new emulator.
[2009-03-21 11:28:32 - Hello2] Launching a new emulator.
[2009-03-21 11:28:33 - Emulator] ### WARNING: Another emulator is running with our data file
[2009-03-21 11:28:33 - Emulator] ### WARNING: User data changes will NOT be saved!
[2009-03-21 11:28:34 - Hello2] New emulator found: emulator-5556
[2009-03-21 11:28:34 - Hello2] Waiting for HOME ('android.process.acore') to be launched...
[2009-03-21 11:28:35 - Emulator] 2009-03-21 11:28:35.818 emulator[26688:10b] Warning once: This
[2009-03-21 11:29:36 - Hello2] HOME is up on device 'emulator-5556'
[2009-03-21 11:29:36 - Hello2] Uploading Hello2.apk onto device 'emulator-5556'
[2009-03-21 11:29:36 - Hello2] Installing Hello2.apk...
[2009-03-21 11:29:40 - Hello2] Success!
[2009-03-21 11:29:40 - Hello2] Starting activity com.example.hello.Hello2 on device
[2009-03-21 11:29:43 - Hello2] ActivityManager: Starting: Intent { comp={com.example.hello/com.
    
```

Mortgage calculator in Android

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_mtg);
    principal = (EditText) findViewById(R.id.principal);
    payment = (EditText) findViewById(R.id.payment);
    rate = (EditText) findViewById(R.id.rate);
    final Button compute = (Button) findViewById(R.id.compute);
    final Button clear = (Button) findViewById(R.id.clear);
    myListView = (ListView) findViewById(R.id.listView1);
    compute.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            rows = pay();
            msgbox("rows.size() = " + rows.size());
            extrarows.clear();
            for (int i = 0; i < rows.size(); i++) {
                extrarows.add(rows.get(i)); // no idea why two arrays
            }
            aa.notifyDataSetChanged();
        }
    });
}
```

Mortgage calculator in Android (p 2)

```
clear.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        extrarows.clear();
        aa.notifyDataSetChanged();
    }
});
principal.requestFocus(); // maybe not needed

rows.add("1"); rows.add("two"); // verify life
int layout = R.layout.my_list_item_1;
//int layout = android.R.layout.simple_list_item_1;
aa = new ArrayAdapter<String>(this, layout, extrarows);
myListView.setAdapter(aa);
}
```


Mortgage calculator in Android (p 3)

The screenshot displays the Android Studio IDE with the following components:

- Project Structure:** Shows the project hierarchy for 'Mtg' (app, src, main, res, layout) and 'activity_mtg.xml'.
- XML Editor:** Displays the layout XML for 'activity_mtg.xml' in Text mode. The code defines a `RelativeLayout` containing three `EditText` views and a `ListView`.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context=".MtgActivity" >

    <EditText
        android:id="@+id/principal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:hint="principal"
        android:ems="10" >

        <requestFocus />
    </EditText>

    <EditText
        android:id="@+id/payment"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/principal"
        android:layout_below="@+id/principal"
        android:hint="payment"
        android:ems="10" />

    <EditText
        android:id="@+id/rate"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/principal"
        android:layout_below="@+id/payment"
        android:hint="rate %"
        android:ems="10" />

    <ListView
        android:id="@+id/list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:padding="10dp"
        android:divider="@android:drawable/divider_horizontal_textview"
        android:dividerHeight="10dp"
        android:choiceMode="singleChoice"
        android:layout_marginTop="20dp" />
</RelativeLayout>
```
- Preview:** Shows a visual representation of the app on a Nexus 4 device. The interface includes input fields for 'principal', 'payment', and 'rate %', 'Compute' and 'Clear' buttons, and a list of items (Item 1 to Item 5).
- Bottom Bar:** Contains tabs for Terminal, Messages, Android, Run, TODO, Event Log, Gradle Console, and Memory Monitor. A status bar at the bottom indicates 'Gradle build finished in 4 sec (14 minutes ago)'.

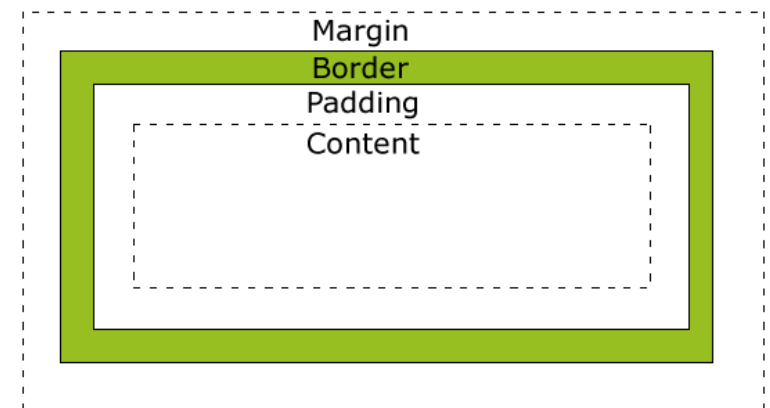
HTML Layout

- mostly based on `<div>` tags, suitably nested:

w3schools.com



- any element has these properties:



- use *CSS* styles for width, float, etc., to adjust sizes, control position

HTML

```
<div style="width:100%"> <!-- top part -->
  <form>
    Principal <input type=text size=10 class=in id=prin value="10000">
    Interest Rate <input type=text class=in id=rate value="5">
    Monthly Payment <input type=text class=in id=mp value="100">
  </form>
</div> <!-- end top part →
<div style="clear:both"> Payment Schedule: </div>
<div style="float:left"> <!-- schedule -->
  <form style="float:left">
    <textarea id="sched" cols=80 rows=20 style="background-color:#ffe;
      height:300px"> </textarea>
  </form>
</div> <!-- end schedule →
<div style="width:100px; height:300px; float:left;
  background-color:#eee"> <!-- buttons -->
  <form>
    <input type=button value="Update" class=but onClick="xupdate()"> <br>
    <input type=button value="Clear" class=but onClick="xclear()"> <p>
    <input type=button value="Quit" class=but onClick="xquit()">
  </form>
</div> <!-- end buttons -->
```

Javascript

```
function xupdate() {
    var mp = parseFloat(document.getElementById("mp").value);
    var rate = parseFloat(document.getElementById("rate").value);
    var prin = parseFloat(document.getElementById("prin").value);
    var sched = document.getElementById("sched");
    var totint = 0, totprin = 0;
    for (i = 1; i <= 500; i++) {
        xint = prin * rate/12.0/100.0;
        var dp = mp - xint;
        if (prin - dp > 0) {
            prin -= dp;
        } else {
            dp = prin;
            prin = 0;
        }
        str = i + "\t\t" + xint.toFixed(2) + "\t\t" + prin.toFixed(2) + "\t\t" + dp.toFixed(2);
        if (i == 1)
            sched.value = str;
        else
            sched.value += str;
        totint += xint;
        totprin += dp;
        if (prin <= 0)
            break;
    }
}
```

Tcl/Tk

- **Tcl: tool command language**
 - scripting language
 - extensible by writing C functions
- **Tk: (windowing) toolkit**
 - widget set for graphical interfaces
 - (IMHO) the best widget set ever
- **created by John Ousterhout**
 - Berkeley, ~1990
 - see www.tcl.tk

- **Tk is embedded in other languages**
 - TkInter in Python
 - Perl/Tk
 - Ruby
 - ...

Tcl example

- name-value addition

```
while { [gets stdin line] > -1 } {
    scan $line "%s %s" name val
    if {[info exists tot($name)]} {
        incr tot($name) $val
    } else {
        set tot($name) $val
    }
}

foreach i [array names tot] {
    puts "[format {%10s %4d} $i $tot($i)]"
}
```

Tcl example 2: formatter

```
set space ""; set line ""

proc addword {w} {
    global line space
    if {[expr [string length $line] + [string length $w]] > 60} {
        printline
    }
    set line "$line$space$w"
    set space " "
}

proc printline {} {
    global line space
    if {[string length $line] > 0} {
        puts $line
    }
    set line ""; set space ""
}

while {[gets stdin in] >= 0} {
    if {[string length $in] > 0} {
        for {set i 0} {$i < [llength $in]} {incr i} {
            addword [lindex $in $i]
        }
    } else {
        printline
        puts "\n"
    }
}
printline
```

Hello world in TkInter & Ruby

- Python

```
from Tkinter import *
root = Tk()
frame = Frame(root)
frame.pack()
button = Button(frame, text="hello world", command=frame.quit)
button.pack()
root.mainloop()
```

- Ruby

```
require 'tk'
root = TkRoot.new { }
TkButton.new(root) do
  text "hello world"
  command { exit }
  pack()
end
Tk.mainloop
```


Mortgage calculator in Tcl/Tk

```
frame .ent
entry .ent.principal -text "Principal"
entry .ent.payment -text "Payment"
entry .ent.rate -text "Rate %"
pack append .ent .ent.principal left \
               .ent.payment left .ent.rate left

frame .but
button .but.compute -text Compute -command "compute"
button .but.clear -text Clear \
               -command ".txt.t delete 1.0 end"
pack append .but .but.compute left .but.clear left

frame .txt
text .txt.t -yscrollcommand ".txt.s set"
scrollbar .txt.s -command ".txt.t yview"
pack append .txt .txt.t {left expand} .txt.s {right fillly}

pack append . .ent top .but top .txt expand
```

Mortgage calculator in Tcl/Tk

```
proc compute {} {
    set prin [.ent.principal get]
    set pay [.ent.payment get]
    set rate [expr [.ent.rate get]/12/100.0]

    set totint 0.0
    set totprin 0.0
    for {set i 1} {$i <= 500} {incr i} {
        set Int [expr $prin * $rate]
        set dp [expr $pay - $Int]
        if {[expr $prin - $dp] > 0} {
            set prin [expr $prin - $dp]
        } else {
            set dp $prin
            set prin 0
        }
        .txt.t insert end [format "%d\t%.2f\t%.2f\t%.2f\n" $i $Int $dp $prin]
        set totint [expr $totint + $Int]
        set totprin [expr $totprin + $dp]
        if {$prin <= 0} { break }
    }
    .txt.t insert end [format "\t\t%.2f\t%.2f\n" $totint $totprin]
}
```