

Graphical user interfaces

- **interfaces are 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**
 - some programmable, some purely by drawing
 - may adapt to changes like reshaping
- **Swing package (javax.swing):**
 - runs standalone everywhere, can be used on web pages in applets
 - Google Web Toolkit is similar
- **other GUI systems are analogous, but with many differences**

Swing examples

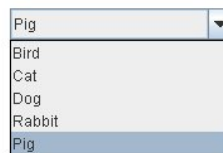
<http://java.sun.com/docs/books/tutorial/uiswing/index.html>



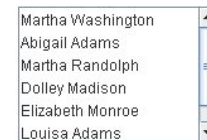
[JButton](#)



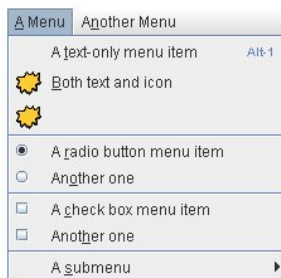
[JCheckBox](#)



[JComboBox](#)



[JList](#)



[JMenu](#)



[JRadioButton](#)



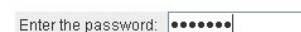
[JSlider](#)



[JSpinner](#)



[JTextField](#)



[JPasswordField](#)

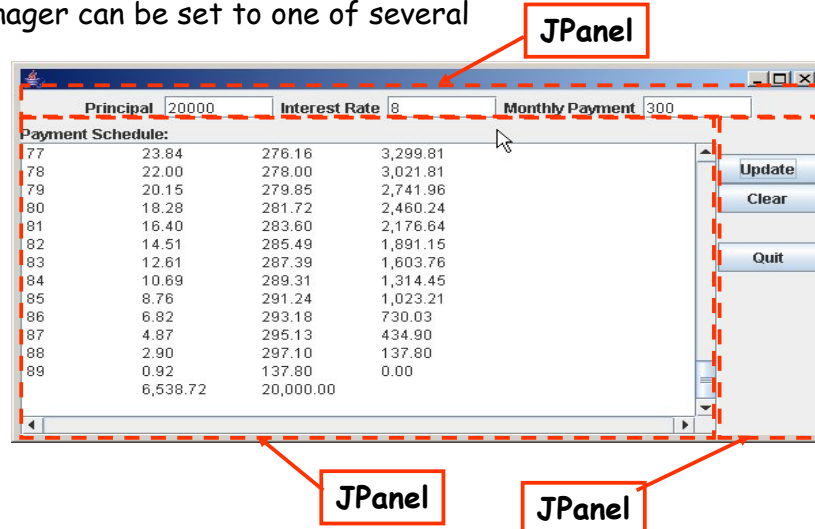
Component object hierarchy

```
Object
  Component
    Container
      JComponent
        JPanel
        JLabel
        JButton
        JTextComponent
          JTextField
            JFormattedTextField
            JPasswordField
          JTextArea
          JEditorPane
          JTextPane
```

- **containers hold components & containers, used to build up nested structure:**
 - 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, JText, ...**
 - 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



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 in Swing

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

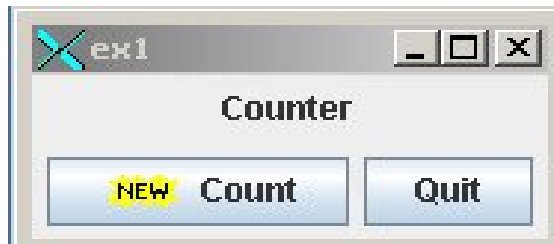
```
    JButton jb = new JButton("whatever");  
    jb.addActionListener(this);
```

 - 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**

```
JLabel label = new JLabel("0");
JButton button = new JButton("Lookup");
button.addActionListener(
    new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            n++;
            label.setText(n);
        }
    }
);
```

- **equivalent to this, without separate declaration and name**

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

Layout manager approaches

- **Java**
 - position by imperative code, with 8 standard layout managers
 - graphical layout by NetBeans IDE
- **Tk**
 - mostly declarative: position relative to other positioned objects
- **VB (pre .NET)**
 - mostly draw on a screen: absolute positioning
 - can modify dynamically by setting properties
- **C#**
 - 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

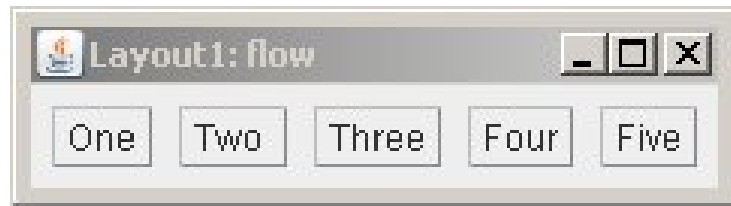
Layout managers in Swing

- **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
(`PAGE_START`, `PAGE_END`, `LINE_START`, `CENTER`, `LINE_END`)
- **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);
    }
}
```

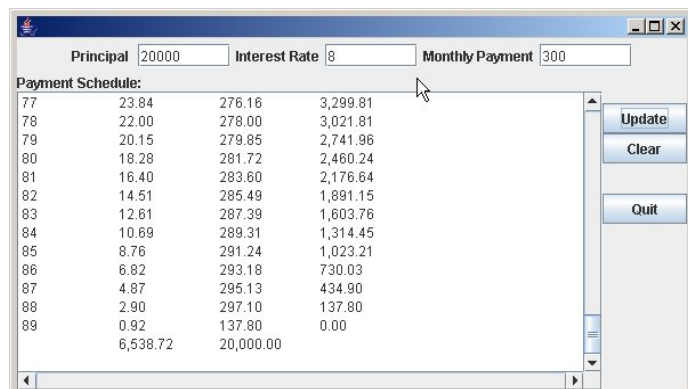
Grid Layout

```
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);
    }
}
```



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.



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

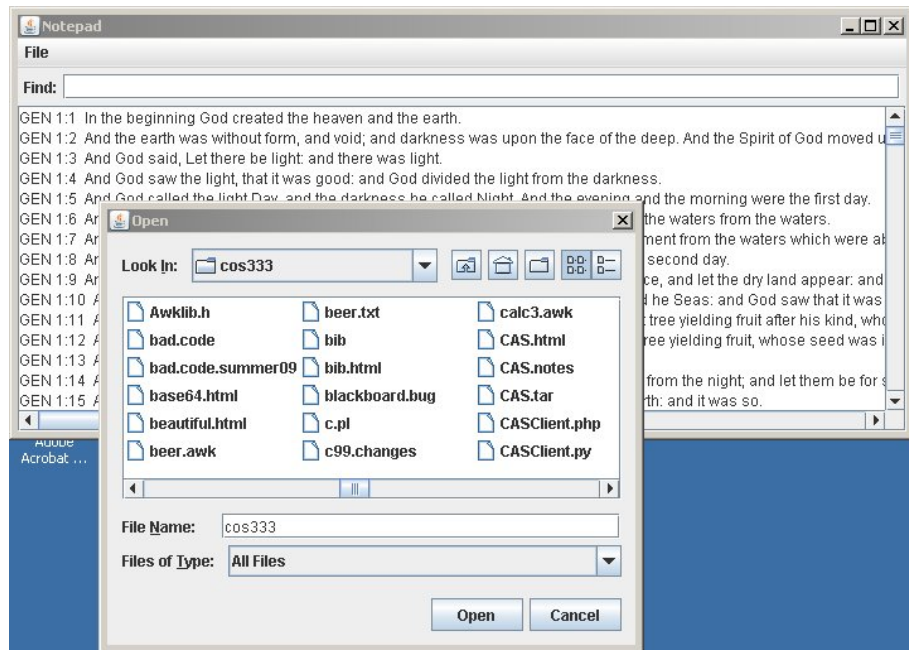
```
String pay() {
    double mp = Double.parseDouble(tmpay.getText());
    double prin = Double.parseDouble(tprin.getText());
    double mrate = Double.parseDouble(trate.getText())/12/100;
    double totint = 0, totprin = 0;
    String s = "";
    for (int i = 1; i <= 500; i++) {
        double Int = 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;
}
```

Google Web Toolkit version

```
public class Mtg implements EntryPoint {
    public void onModuleLoad() {
        final Label lprin = new Label("Principal");
        final TextBox tprin = new TextBox();
        final Label lrate = new Label("Interest Rate");
        final TextBox trate = new TextBox();
        final Label lmpay = new Label("Monthly Payment");
        final TextBox tmpay = new TextBox();
        final Label lsched = new Label("Monthly Payment");
        final TextArea tpay = new TextArea();
        tpay.setCharacterWidth(45); tpay.setVisibleLines(15);
        final Button update = new Button("Update");
        final Button clear = new Button("Clear");
        final Button quit = new Button("Quit");
        final ScrollPanel jsp = new ScrollPanel(tpay);
        update.addClickListener(new ClickListener() {
            public void onClick(Widget sender) {
                tpay.setText(pay(tmpay, tprin, trate));
            }
        });
        clear.addClickListener(new ClickListener() {
            public void onClick(Widget sender) { tpay.setText(""); }
        });
        quit.addClickListener(new ClickListener() {
            public void onClick(Widget sender) { tpay.setText("no escape yet!");
        });
        RootPanel.get("slot1").add(update);
        RootPanel.get("slot1").add(clear);
    }
}
```

Example 3: Notepad editor

- menus
- file dialog
- I/O from file system



Notepad, page 2

```
public class Notepad extends JFrame implements ActionListener {
    int last;
    JLabel lfind = new JLabel("Find:");
    JTextField tf = new JTextField(60);
    JTextArea ta = new JTextArea(15,60);
    JMenuBar mb = new JMenuBar();
    JMenu mfile;

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

    Notepad() {
        setTitle("Notepad");
        setJMenuBar(mb);
        mfile = new JMenu("File");
        JMenuItem mi;
        mfile.add(mi = new JMenuItem("Open"));
        mi.addActionListener(this);
        mfile.add(mi = new JMenuItem("Save"));
        mi.addActionListener(this);
        mfile.add(mi = new JMenuItem("Quit"));
        mi.addActionListener(this);
        mb.add(mfile);
        tf.addActionListener(this);
    }
}
```

Notepad, page 3

```
getContentPane().setLayout(new BorderLayout());
JPanel top = new JPanel();
top.add(lfind); top.add(tf);
getContentPane().add(top, BorderLayout.NORTH);
JScrollPane jsp = new JScrollPane(ta,
    JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
    JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
getContentPane().add(jsp, BorderLayout.CENTER);
pack();
setVisible(true);
setDefaultCloseOperation(EXIT_ON_CLOSE);
}
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == tf) {
        find(tf.getText());
    } else if (e.getSource() instanceof JMenuItem) {
        String b = e.getActionCommand();
        if (b.equals("Quit")) {
            System.exit(0);
        } else if (b.equals("Open")) {
            openfile();
        } else if (b.equals("Save")) {
            savefile(ta.getText());
        }
    }
}
```

Notepad, page 4

```
public void find(String pat) { // find next pat in ta
    String s = ta.getText(); // where to search
    if (last + pat.length() >= s.length())
        last = 0;
    int n = s.indexOf(pat, last); // look to end
    if (n == -1) {
        last = 0;
        n = s.indexOf(pat, 0); // look from beginning
    }
    if (n >= 0) {
        int m = n + pat.length();
        System.err.println("found " + tf.getText() +
            " at " + n + ", " + m);
        //ta.setHighlighter(new DefaultHighlighter());
        //ta.setSelectedTextColor(Color.green);
        ta.setSelectionColor(Color.red);
        ta.getCaret().setSelectionVisible(true);
        ta.select(n, m);
        last = n + 1;
    }
}
```

Notepad, page 5

```
public void openfile() {
    JFileChooser jfc = new JFileChooser();
    jfc.showOpenDialog(this);
    if (jfc.getSelectedFile() == null)    // cancelled
        return;

    File fil = jfc.getSelectedFile().getAbsoluteFile();
    String f = fil.getAbsolutePath(); // attach directory name
    try {
        FileInputStream in = new FileInputStream(f);
        byte [] data = new byte [in.available()];
        in.read(data);
        ta.setText(new String(data));
    } catch (IOException e) {
        ta.setText("Can't open file " + f);
    }
}
```

Notepad, page 6

```
public void savefile(String s) {
    JFileChooser jfc = new JFileChooser();
    jfc.showSaveDialog(this);
    if (jfc.getSelectedFile() == null)    // cancelled
        return;

    File fil = jfc.getSelectedFile().getAbsoluteFile();
    String f = fil.getAbsolutePath(); // attach directory name
    try {
        FileOutputStream out = new FileOutputStream(f);
        out.write(s.getBytes());
        out.close();
    } catch (FileNotFoundException e) {
        System.err.println(e + " can't open " + f);
    } catch (IOException e) {
        System.err.println(e + " savefile error");
    }
}
```

Applets

- **run Java code in browser**
- **HTML applet tag loads code into a web page**

```
<applet  
    code="scribble.class" width=500 height=300>  
</applet>
```
- **default security restrictions on applets**
 - can't access client file system
 - can't run processes on client
 - can't create unrestricted top-level windows
 - can't make unrestricted network connections
 - can't make the interpreter quit
 - etc., etc.
- **can use Socket() in an applet**
 - but can only open socket to system that applet came from