

Dialog use

```
// application
construct dialog
bind event to event dialog
  handler
....
invoke dialog event handler
returnState = showDialog(...)
....
// application method waits
switch returnState {
  // process result
  // of dialog
}
```

```
Dialog displayed
user interact w/
  dialog
hide dialog
dialog not displayed
```

Dialog object can invoke show / hide multiple times
values preserved across displays

<< see CustomDialog, DateChooser >>

JDialog

Custom dialog class extend JDialog.

Design of the containment hierarchy is same as for JFrame.

Create an instance of the dialog before it will be used.

JDialog(Frame owner, String title, boolean modality)

in dialog's constructor

super(owner, title, modality);

use a modal dialog? must dialog be dealt with by user ?

setResizable(false) usually dialogs are not change by user

keep **HIDE_ON_CLOSE** as defaultCloseOperation

write accessor methods to get values from dialog object

JDialog inherits a **show()** and **hide()** method from Dialog.

show() will block until modal dialog is closed

can provide a "showDialog(...)" method that positions dialog relative to owner and returns state of how dialog was closed

hide() will remove dialog from display (but not dispose of it).

JApplet

Swing GUIs can be developed as applications extending JFrame, applets extending JApplet, or both!

To create an applet your class should extend JApplet and all your content construction should be done inside method

```
public void init() {...}
```

You do not need a main() method or an explicit constructor. The code you would put in the constructor should be put in init().

The default layout manager for JApplet content pane is BorderLayout. Menus can be added to JApplet with **setJMenuBar (aMenu)**

There are several examples of applets in the Dietel book, although few use many Swing classes.

Applet tag

You will also need to create an html file to embed the applet class.

You run your applet by using Sun's appletviewer with the html file or your browser.

You may have to use Sun's htmlconverter to convert your html file to run your applet.

at dos prompt enter htmlconverter and then follow dialog choices.

```
<html> <body>
<applet code = "applet.class" codebase = "path"
width = "n" height = "m" >
</applet>
</body> </html>
```

Combined application and applet code in same application.

JFrame & JApplet load images differently -- remember start type

In PipeApplet.java I set menu in BorderLayout.NORTH ...

JFrame, JDialog, JApplet can all use setJMenuBar()

Application || Applet

PipeApplet.java can be run as either as an applet or an application.

```
public class AppAppletClass extends JApplet {
    boolean asApplet;
    ...
    public void init() {
        asApplet = true;
        setContentPane(makeContentPane()); }

    public static void main() {
        asApplet = false;
        JFrame frame = new JFrame("title");
        AppAppletClass applet = new AppAppletClass();
        frame.setContentPane(applet.makeContentPane());
        // setBounds, setVisible, setDefaultCloseOperation.... }

    public Container makeContentPane() {
        Container contentPane = getContentPane();
        // menus should be added to BorderLayout.NORTH
        // all content construction of GUI controls ...
        return contentPane; }
    ... }
```

JComponent

Components are UI elements that are the members of containers and managed by layout managers. They are the *visible* elements of the UI.

Common Components

controls: JButton, JTextField, JSlider

displays: JLabel, JToolTip, JProgressBar

All components have many properties for location, size, color, ... some have an visually interesting properties like icon.

Properties manipulated with set* and get* methods

```
aLabel.setText(new String("Read me!"));  
aString = aTextField.getText();  
aLabel.setFont(new Font("Sans-Serif", Font.PLAIN, 24));  
aButton.setBackground(Color.blue);  
aLabel.setIcon(new ImageIcon("anIcon.gif"));  
aLabel.setMinimumSize(new Dimension(width, height));  
aButton.setCursor(new Cursor(Cursor.HAND_CURSOR));
```

JLabel, JButton

```
JLabel("Hi There");    JLabel("hi", JLabel.RIGHT);  
JLabel(new ImageIcon("icon.gif"));  
JLabel("An icon", new ImageIcon("icon.gif"),  
      JLabel.Center);
```

HTML can be used to specify Strings in labels, buttons, tooltips, ...

JButton

```
... implements ActionListener ...  
JButton b1 = new JButton("Button 1");    // b2 also ...  
b1.setMnemonic("1");                    // "alt 1"    // b2 also  
b1.setEnabled(false);                   // initially disabled  
b1.addActionListener(this);             // b2 also ...  
...  
public void actionPerformed(ActionEvent e) {  
    if (e.getSource() == b2)    {  
        b1.setEnabled(true);  
        b2.setEnabled(false); } ...
```

Group of buttons / options

check box none, one, or many can be selected

radio button, none or one option selected (mutually exclusive)

```
...
int i, state[] = {0,0,0,0}; // java has no set type
JPanel checkPanel = new JPanel(new GridLayout(0,1));
CheckBoxListener ckListener = new CheckBoxListener();
JCheckBox b[] = new JCheckBox[4];
for(i=0; i<4; i++) {
    b[i] = new JCheckBox(Integer.toString(i));
    b[i].addItemListener(ckListener);
    checkPanel.add(b[i]);}
...
```

```
class CheckBoxListener implements ItemListener) {
    public void itemStateChanged(ItemEvent e) {
        Object source = e.getItemSelectable
        for(i=0; i<4; i++)
            if (source == b[i]) state[i] = 1;
            else state[i] = 0; .... }
}
```

Java Graphics

Component and Graphics classes (also Java2D API and Java 3D API)

Drawing graphics has 2 parts / steps

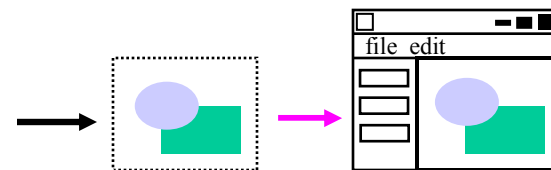
How to draw -- attributes of the graphic context (Graphics)

Color, Font, foreground, background, ...

What to draw -- a draw request

drawLine(...), drawRect(...), fillRect(...), ...

Drawing differs in AWT & Swing



AWT heavyweight components are drawn directly -- not double buffered. This can cause flicker w/ animation & requires double buffering techniques.

Swing lightweight components are double buffered (drawn indirectly) if they are held by JPanel or JRootPane containers.

Otherwise they must be set to be set to require double buffering

<http://java.sun.com/docs/books/tutorial/extra/fullscreen/doublebuf.html>

Graphics

Write a JComponent's `paintComponent(Graphics g)` method – or `paint(Graphics g)` – for custom drawing. (`paintComponent` can call your own draw functions passing `Graphics g...`)

```
public void paintComponent (Graphics g) {  
    g.setColor(new Color (255, 0, 0));  
    g.fillRect(25, 25, 100, 20);  
    g.drawString("rgb is:  " + g.getColor(), 130, 40);  
    g.setColor(new Color (0.0f, 1.0f, 0.0f));  
    g.fillRect(25, 50, 100, 20);  
    g.drawString("rgb is:  " + g.getColor(), 130, 40);  
}
```

Usually `Graphics g` is not known, to call `paint(g)` directly...

To invoke `paintComponent` call `repaint()`;

`Repaint` invokes `update()`. // `Update()` called on `expose` & `configure`

`Update` clears the component's background of any previous draws and calls `paint(...)`.

Override `update()` to implement double buffering in AWT.

Swing drawing

invoke `super.paintComponent()` in overridden `paintComponent` to have UI delegate clear background for opaque components.

calls to `repaint()` invoke the following methods

`JComponent`'s `update()` calls `paint()` directly

`JComponent`'s `paint()` is call (in this order)

`paintComponent` -- custom painting

`paintBorder` -- insures border is painted

`paintChildren` -- `JComponent` can be a container ...

Sometimes `repaint()` has to be called after `revalidate()` when `revalidate` does not force re-management of container's children.

Java 2D API more shapes, line styles, `bufferedImages`

Java 3D API 3D retained graphics, scene graph based API.

Draw requests

All public void. All arguments int unless specified.

```
drawLine( x1, y1, x2, y2);
drawRect( x, y, width, height);
fillRect(...);
clearRect(...); // draws in background
drawRoundRect( x, y, width, height, arcWidth, arcHeight);
fillRoundRect(...);
draw3DRect(x, y, width, height, boolean raised);
drawOval(x, y, width, height);
fillOval(x, y, width, height);
drawArc(x, y, width, height, startAngle, arcAngle);
    angles are degrees.
fillArc(...);
drawPolygon(xPoints[], yPoints[], points); // closed poly
drawPolyLine(xPoints[], yPoints[], points);
drawFillPolygon(xPoints[], yPoints[], points); // closed

public Polygon(xValues[], yValues[], points);
drawPolygon(Polygon p);
fillPolygon(Polygon p);
```

Color

```
Color(int r, int g, int b)           // 0..255
Color(float r, float g, float b);    // 0.0f .. 1.0f

static fields of Color:
Color.orange, pink, cyan, magenta, yellow, black,
white, gray, light gray, dark gray, red, green, blue

Color.orange    Color(255, 200, 0)
Color.pink      Color(255, 175, 175)

public void paintComponent(Graphics g) {
    g.setBackground(Color.black);
    g.setColor = new Color(120, 60, 230); // ?color?
    g.drawLine(10, 10, 100, 100);
    ... }
```

Using JColorChooser to get a color

```
...
color = JColorChooser.showDialog(...);
if (color == null) color = Color.black; // cancel state
aComponent.setBackground(color);
aComponent.repaint();
```

Fonts

Text is hard in bit map graphic systems, graphics is easy.

```
Font(String familyName, int style, int size);
g.setFont(new Font("Serif", Font.PLAIN, 18);
```

family: Courier, Dialog, DialogInput, Helvetica,
 TimesRoman, Symbol

styles: PLAIN, BOLD, ITALIC, ITALIC+BOLD

The FontMetric class stores rendering information about the font family:

```
FontMetric fm = g.getFontMetric(aFont);
```

Available fonts on the system can be obtained.

```
GraphicsEnvironment ge =
    GraphicsEnvironment.getLocalGraphicsEnvironment();
Font[] fontArray = ge.getAllFonts(); // all size 1
```

New fonts can be derived from existing fonts: (see fontList.java)

```
aFont = tFont.deriveFont(float size); // or int style
aFont = tFont.deriveFont(int style, float size);
```

Mouse Event Handling: MouseAdapter, MouseMotionAdapter

MouseEvent methods:

```
int getClickCount(); // count quick consecutive clicks
int getX(), int getY(); // get point position values
Point getPoint();
boolean isPopupTrigger();
Component getComponent();
int getWhen(); // returns time stamp
boolean isAltDown(); isMetaDown(); isShiftDown();
```

static SwingUtilities

```
static boolean isLeftMouseButton(MouseEvent);
static boolean isMiddleMouseButton(MouseEvent);
static boolean isRightMouseButton(MouseEvent);
```

MouseListener

mousePressed(MouseEvent)
mouseClicked(MouseEvent)
mouseReleased(MouseEvent)
mouseEntered(MouseEvent)
mouseExited(MouseEvent)

button is pressed over component
button released after press w/o drag
button is released
mouse enters component area
mouse leaves component area

MouseMotionAdapter

mouseDragged(MouseEvent)
mouseMoved(MouseEvent)

mouse pressed and dragged
mouse moved (not pressed)

(see Hello585Swing.java example)

Keyboard Event Handling

KeyListener has methods for keyPressed, keyReleased, keyTyped, isMetaDown(), isAltDown

Useful in JTextArea objects

Actions enable several UI components that perform same action (say menu item and toolbar) and process text, icon and state of the menu item and toolbar -- in one place. << see text's ActionDemo >>

```
Action leftAction = new AbstractAction (
    "Go left", new ImageIcon("../left.gif")) {

    public void actionPerformed(ActionEvent e) {
        displayResult("Action for 1st button, menuItem", e);
    }
}
.....
JButton button = toolBar.add(leftAction);
JMenuItem menuItem = aMenu.add(leftAction);
....
// disable (or enable w/ true) actions
leftAction.setEnabled(false);
```

Timer objects fire action events after a specified delay.

tooltips, autoscrolling, and progress bar are classes that use timers
animation of images

on-going task (clocks, monitoring, drawing) and user interaction.

Timers can inject an action into the on-going event dispatching thread (the run of your application).

```
public static void main(String args[]) {  
    SwingApp mainW = new SwingApp("Swing App Title");  
    mainW.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    mainW.setBounds(20,20,400,200);  
    mainW.setVisible(true); // realizes frame  
} // main thread is not active now, event thread is.
```

```
Timer ( int delayMsecs, ActionListener timerAction);
```

```
Timer blinkTimer = new Timer(1000, new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        if(blinkButtonOn)  
            blinkButton.setText("");  
        else  
            blinkButton.setText("Blink!");  
        blinkButtonOn = !blinkButtonOn; }    });
```

timers default to repeat counting down and firing action events

```
boolean isRunning();  
void setDelay(int);    int getDelay();  
void start(); // turn timer on.  
void stop();
```

Threads

Swing components can be accessed by one thread at a time (usually the event-dispatching thread) . Swing **IS NOT** thread safe.

General usage:

Once a component is realized all code related to the component should be executed in the event-dispatching thread.

Exceptions:

repaint() and revalidate() queue method requests in the event dispatching thread – can be called from other threads

add and remove listeners are thread safe

SwingUtilities that allow threads to place code in event-dispatch thread

invokeLater(...) // does not wait for method to be executed

invokeAndWait(...) // can deadlock

<< see ThreadDemo.java Note dialog modality not thread based. >>