Swing コンポーネントの構成

http://feather.cocolog-nifty.com/weblog/2007/10/swing_swing2_ro_34ad.html
Swing のフレームを構成するレイヤーの構成は以下のとおり。

エンターキーでボタンを選択する

```
KeyStroke enterPress = KeyStroke.getKeyStroke(KeyEvent.VK_ENTER, 0, false);
KeyStroke enterRelease = KeyStroke.getKeyStroke(KeyEvent.VK_ENTER, 0, true);

button.getInputMap().put(enterPress, "pressed");
button.getInputMap().put(enterRelease, "released");
```

フォントのアンチエイリアス

http://itpro.nikkeibp.co.jp/article/COLUMN/20070205/260649/
http://d.hatena.ne.jp/itiri/20080224/1203856116

■ Java6 の場合

```
awt.useSystemAAFontSettings
```

オプションを使う。
設定出来る値は

--	--
off	アンチエイリアスを行わない
on	アンチエイリアスを行う
gasp	小さい文字ではアンチエイリアスを行わない
lcd	サブピクセルのアンチエイリアス
lcd_hrgb	サブピクセルのアンチエイリアス 横方向 RGB
lcd_hbgr	サブピクセルのアンチエイリアス 横方向 BGR
lcd_vrgb	サブピクセルのアンチエイリアス 縦方向 RGB
lcd_vbgr	サブピクセルのアンチエイリアス 縦方向 BGR

デフォルトでは、gasp になっているみたい。

例（実行時）：

```
java -Dawt.useSystemAAFontSettings=on test.class
```

例（プログラム内）：

```
System.setProperty("awt.useSystemAAFontSettings","on");
```

■ Java5 の場合

```
swing.aatext
```

オプションを使う。
例（実行時）：

```
java -Dswing.aatext=true test.class
```

例（プログラム内）：

```
System.setProperty("swing.aatext","true");
```

Look and Feel を変更する

http://terai.xrea.jp/Swing/LookAndFeel.html

```
private static final String mac     = "com.sun.java.swing.plaf.mac.MacLookAndFeel";
private static final String metal   = "javax.swing.plaf.metal.MetalLookAndFeel";
private static final String motif   = "com.sun.java.swing.plaf.motif.MotifLookAndFeel";
private static final String windows = "com.sun.java.swing.plaf.windows.WindowsLookAndFeel";
private static final String gtk     = "com.sun.java.swing.plaf.gtk.GTKLookAndFeel";
private static final String nimbus  = "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel";
private void setLookAndFeel(String laf) {
  try{
    UIManager.setLookAndFeel(laf);
    //SwingUtilities.updateComponentTreeUI(frame);
  }catch(Exception ex) {
    ex.printStackTrace();
  }
}
```

フォントを一括で変更する

http://guijava.180r.com/p%A5%D5%A5%A9%A5%F3%A5%C8%A4%F2%B0%EC%B3%E7%CA%D1%B9%B9%A4%B9%A4%EB.html

```
Font font = new Font("SansSerif",Font.PLAIN,18);
LookAndFeel laf = UIManager.getLookAndFeel();
Set keys = laf.getDefaults().keySet();
for (Iterator it = keys.iterator(); it.hasNext();) {
 String key=it.next().toString();
 if (key.indexOf("font") != -1) {
   UIManager.put(key, font);
 }
}
```

JDialog などからタイトルバーなどの枠を消す

```
dialog.setUndecorated(true);
```

とすると、タイトルバーなどが無い JDialog が作成できる。

JFrame を中央に表示する

http://terai.xrea.jp/Swing/CenterFrame.html

```
JFrame frame = new JFrame(" フレームをスクリーン中央に表示 ");
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
frame.getContentPane().add(new MainPanel());
frame.pack();
frame.setLocationRelativeTo(null);
// 以下は自前で位置を計算する場合
//Rectangle screen = frame.getGraphicsConfiguration().getBounds();
```

```
//frame.setLocation(screen.x + screen.width/2  - frame.getSize().width/2,
//                  screen.y + screen.height/2 - frame.getSize().height/2);
frame.setVisible(true);
```

デフォルトボタン

どこにフォーカスがあっても、エンターキーを押されたときに反応する
ボタン（デフォルトボタン）を設定するには

```
JRootPane#setDefaultButton(JButton defaultButton)
```

を使う。

JLabel の色が親パネルと同じになってしまう

```
setOpaque(true)
```

にすると透過しない。逆に

```
setOpaque(false)
```

にすると透過（親パネルと同じ色）になる。

swing コンポーネントで入力制限

Swing コンポーネントで入力制限

swing コンポーネントのイベントを他コンポーネントにスルーさせる

Swing コンポーネントのイベントをスルーさせる

awt イベントが全て終了してから処理をする

```
SwingUtilities.invokeLater
または
Toolkit.getDefaultToolkit().getSystemEventQueue().invokeLate
```

を使う。
基本的にどちらも同じイベントディスパッチスレッドにイベントを追加する・・・はず。

awt イベントの終了を待つ

```
 SwingUtilities.invokeAndWait
または
Toolkit.getDefaultToolkit().getSystemEventQueue().invokeAndWait
```

を使う。

こんな感じでメソッド作っても良いかも。

```
protected void waitDoEvent(){
   try {
     if (!EventQueue.isDispatchThread()){
       // Thread.sleep(100);  これだけでも良いような気がする・・・。
       SwingUtilities.invokeAndWait(new Runnable(){
```

```
                public void run(){
                }
            });
        }
    } catch (Exception e){}
}
```

ボーダー

■ BorderFactory を使う

javax.swing.BorderFactory を使うと便利。

```
createTitledBorder("hoge") ;
```

とか。

■ javax.swing.border のクラスを使う

javax.swing.border のボーダークラスを使うと色々な線をコンポーネントの周りに描ける。

```
LineBorder.createBlackLineBorder()
```

とか

```
new TitledBorder(LineBorder.createBlackLineBorder() , "タイトル")
```

ボーダーの周りに隙間を空けたい場合は、EmptyBorder を使う。

```
label = new JLabel("ラベルだよ");
Border margin = new EmptyBorder(0,5,0,5);
label.setBorder(new CompoundBorder(LineBorder.createBlackLineBorder(), margin));
```

みたいな感じ。

InputVerifier のタイミング

JComponent を継承しているコンポーネント間でフォーカスを移動使用としたときに、
フォーカスの移動元のコンポーネントの InputVerifier に対して、verifiy が呼ばれる。

よって、フォーカスの移動元、移動先が共に

早い話が Swing コンポーネント以外が画面にいると正常に動かない可能性有り。

モーダルについて

■親子関係ない場合

・モーダルが複数表示された場合、基本的にあとに表示されたほうが前面に出る。

■親子関係がある場合

・子は常に親の前面に表示される。
・親が非表示になると子も非表示になる

Swing コンポーネントの描画について

■ update と repaint

・Swing では、update は呼ばれない。
・repaint イベントディスパッチャ以外から呼んでも良い。（イベントディスパッチャに再描画を依頼するのが repaint メソッド）

■ Swing のペイントの処理過程

Swing が行う " リペイント " リクエストの処理は、AWT とやや違います。ただし、アプリケーションプログラムにとって、その最終結果はどちらも基本的に同じです -- つまり paint() が呼び出されます。Swing は RepaintManager API（後述）を使ってリペイントリクエストを処理し、ペイントの実行性能を上げています。Swing のペイントは、次のような二つの過程をたどります：

(A) ペイントリクエストが最初の親（最外側のウィンドウコンテナ）である重量コンポーネント（通常は JFrame, JDialog, JWindow, または JApplet）に来たとき：

イベントディスパッチスレッド (event dispatching thread, EDT, Java の GUI が動いているスレッド) が、その重量コンポーネントの paint() を呼び出す

Container.paint() のデフォルトの実装が子の軽量コンポーネントの paint() を再帰的に呼び出す

最初の Swing コンポーネントに到着したら、JComponent.paint() のデフォルトの実装が次の処理を行う：

コンポーネントの doubleBuffered プロパティが true で、そのコンポーネントの RepaintManager がダブルバッファリングを on にしていたら、Graphics オブジェクトをオフスクリーンのグラフィクスに変換する .

paintComponent() を呼び出す（ダブルバッファリング on ならオフスクリーンのグラフィクスを渡して）

paintBorder() を呼び出す（ダブルバッファリング on ならオフスクリーンのグラフィクスを渡して）

paintChildren() を呼び出す（ダブルバッファリング on ならオフスクリーンのグラフィクスを渡して）. これはクリップ領域と opaque および optimizedDrawingEnabled を使って、paint() を再帰的に呼び出すべき子のコンポーネントを判断します .

コンポーネントの doubleBuffered プロパティが true で、そのコンポーネントの RepaintManager がダブルバッファリング on なら、元のオンスクリーンの Graphics オブジェクトを使ってオフスクリーンの画像をコピーする .

注 : JComponent.paint() のステップ #1 と #5 は、paint() の再帰的な呼び出し（上のステップ #4 で述べている paintChildren() からの呼び出し）では行われません。ダブルバッファリングでは、一つの同じオフスクリーンイメージを一つのウィンドウ階層中のすべての軽量コンポーネントが共用するからです。

(B) ペイントリクエストが javax.swing.JComponent のサブクラスへの repaint() 呼び出しから来たと

き：

JComponent.repaint() は非同期のリペイントリクエストをコンポーネントの RepaintManager に登録し、RepaintManager は invokeLater() を使って Runnable をキューに入れ、あとでそのリクエストをイベントディスパッチスレッドの上で処理する .

その Runnable オブジェクトがイベントディスパッチスレッドの上で実行されると、コンポーネントの RepaintManager がそのコンポーネントの paintImmediately() を呼び出す . このメソッドは次のことを行う：

クリップ領域と opaque および optimizedDrawingEnabled プロパティを使って、ペイント操作を開始する ‘ ルート ’ のコンポーネントを判断する（透明性とコンポーネントの重なりを正しく扱うため）.

ルートコンポーネントの doubleBuffered プロパティが true で、ルートの RepaintManager の上でダブルバッファリングが on なら、Graphics オブジェクトをオフスクリーンのグラフィクスに変換する .

ルートコンポーネントの paint() を呼び出す（それがさらに、上の (A) の JComponent.paint() のステップ #2-4 を実行する）-- これにより、ルートの中にあってクリップ矩形に含まれるものすべてがペイントされる .

ルートコンポーネントの doubleBuffered プロパティが true で、ルートの RepaintManager の上でダブルバッファリングが on なら、オフスクリーンの画像（オフスクリーンイメージ）を元のオンスクリーンの Graphics オブジェクトを使ってコンポーネントにコピーする .

注 : 一つのコンポーネントやその親のどれかに対して複数の repaint() 呼び出しが行われると、これらの複数の呼び出しが、今 repaint() が呼び出されている最上位（いちばん上の親）のコンポーネントの paintImmediately() への一回のコールバックにまとめられることがあります。たとえば JTabbedPane の中に JTable があって、この収容階層のそのほかのリペイントの処理中に両者が共に repaint() を呼び出すと、この２つの repaint() 呼び出しは JTabbedPane の一回の paintImmediately() 呼び出しとして処理され、そこから、両コンポーネントの paint() が実行されます。

したがって Swing のコンポーネントでは、update() は呼び出されません。

repaint() を呼び出すと paintImmediately() が呼び出されますが、このメソッドはペイント " コールバック " ではないので、ペイントを行うコードを paintImmediately() の中に書いてはいけません。というより、ふつう、paintImmediately() をオーバライドするような状況はほとんどありません。

■ダブルバッファリング

http://www.02.246.ne.jp/~torutk/javahow2/swing.html#doc1_id199
http://wisdom.sakura.ne.jp/system/java/swing/swing8.html
http://homepage1.nifty.com/algafield/paint.html

Swing ではルートコンポーネントの一部として子が描画される。
例えば、JPanel の paint は JPanel の親コンポーネントを辿って行き、ContentPane または、RootPane が JPanel の paint を呼ぶ。

そのため、ダブルバッファを行うための setDoubleBuffered はルートコンポーネントに設定しておけば良い。
基本的に、RootPane も ContentPane も JPanel も標準ではダブルバッファリングは有効になっている。
この時、JPanel#repaint と JFrame#repaint の違いは描画範囲の違いであり、ダブルバッファの設定はルートコンポーネントに左右される。

とりあえず、ダブルバッファを行う場合は、関連コンポーネントを setDoubleBuffered(true) にしておけば良い。

例えば、JPanel をダブルバッファリングで描画する場合
（自前でダブルバッファリングを行うために、Swing のダブルバッファリングを止める場合）
*JPanel#repaint で再描画するときは、RootPane から JPanel まで全てのコンポーネントで setDoubleBuffered(false); する必要がある。
*JFrame#repaint で再描画する場合は、RootPane だけを setDoubleBuffered(false); すれば良い（らしい）。
いずれにしても、関連するものを setDoubleBuffered(false); しておけばいいと思う。

ダブルバッファリングを無効にする場合は、関連コンポーネントの setDoubleBuffered(false) を呼べば良い。

以下の方法でダブルバッファリングを無効にすることもできる。

```
RepaintManager rm = RepaintManager.currentManager(component);
rm.setDoubleBufferingEnabled(false);
```

ただし、各コンポーネントの描画はルートコンポーネントを行う挙動は変わらないので、
JPanel で自作ダブルバッファリングしてもチラツキは無くならないので注意。

キーボードニーモニックの設定

```
setMnemonic(KeyEvent.VK_0);
```

とか

ウインドウを表示したときに、隠れてしまう

ウインドウを表示したときに、隠れてしまうことがある。
Dialog を消す際に、Dialog の親を消すことで Dialog を消したりすると起きるようだ。
きちんと、Dialog の dispose を呼ぶようにする。

メニュー展開の例

Dialog に

```
setUndecorated(true);
```

して、ロード中を画像にしたもの。

JList でスクロールバーの位置を移動させる

JScrollPane で スクロールバーを設定してある JList に対して
スクロールバーを移動したい時は

```
ensureIndexIsVisible
```

を使う

JTable でスクロールバーの位置を移動させる

JScrollPane で スクロールバーを設定してある JTable に対して
スクロールバーを移動したい時は

```
Rectangle rect = table.getCellRect(10,0, false);
scrollpane.getViewPort().scrollRectToVisible( rect );
```

を使う

アニメ gif について

ImageIcon 等に利用する画像に アニメ GIF を利用することができる。
ロード中などの表示に利用すると便利。

ime の表示位置

■ Windows

Windows19 では、
jdk8u131 以降で sun.awt.windows.WInputMethod で エラーが発生するようになった (Google IME の場合)
jdk8u121 までは Windows10 でも問題なし。

■ Linux

https://lists.debian.or.jp/archives/debian-users/201609/msg00005.html

```
 ibus-anthy: ひらがな表示は textfield 内、漢字候補表示は GUI 内の下端左側隅位置
 ibus-mozc: ひらがな表示は textfield 内、漢字候補表示は GUI の下端左側のほぼ外側位置
 fcitx-anthy: GUI の下端左側のほぼ外側位置に表示ブロックが形成され、かつその内側に矩形ラインが形成され、当該矩形ラインの内側にひらがな表示が行われ、space キーを押すと、別途表示ブロック及びその内側の矩形ラインが形成され、当該矩形ラインの内側に漢字候補表示が行われる。
 fcitx-mozc: ひらがな、漢字の何れも共に、GUI の下端左側のほぼ外側位置に出来る大きな表示ブロック内に表示される。
 uim-anthy: ひらがな表示は GUI の下端左側のほぼ外側位置、漢字候補表示は当該ひらがな表示位置の下側位置
 uim-mozc: ひらがな表示は GUI の下端左側のほぼ外側位置、漢字候補表示は当該ひらがな表示位置の下側位置
```

フォントが荒い

環境によってフォントのアンチエイリアスが荒いときがある。
ディスプレイの解像度が高い場合に、OS で拡大率を指定している場合に JavaFX のフォントが荒くなっているように思う。
起動オプションで以下のように倍率を指定するとアンチエイリアスがきれいになる。倍率は 1.0

とか 1.5 とか 2.0 のように指定する

```
-Dsun.java2d.uiScale=1.0
```