Swingによる動的インターフェース設計

Swing APIの外部領域への旅

Peter Seebach
Writer
自由職业者

2006年 4月 25日

Swing UIツールキットは、常に簡単ではありませんが、イベントまたはユーザー・アクションに応じてユーザー・インターフェースを動的に更新することを可能にします。この記事では、動的に更新するUIを構築するための一般的な方法と、その過程で遭遇する可能性のあるいくつかの落とし穴を解説します。また、正しい手法が何であるかを判断するための原則について解説します。

Swingツールキットは、ユーザー・インターフェースを作成するための多彩なツールと、プログラムのライフタイム中にプログラムのインターフェースを変更するための非常に多くのオプションを提供します。これらの機能を注意深く利用すると、ユーザーのニーズに動的に対応し、対話を単純化するインターフェースができあがります。同じ機能を不注意に使用すると、非常にわかりにくく、まったく使用できないプログラムになることもあります。この記事では、動的UIのテクノロジーと概念を紹介し、効果的なUIを構築するヒントを挙げます。Sun JDKに付属のSwingSet2デモ・アプリケーションに基づいて（「参考文献」参照）、ソース・コードを修正します。このアプリケーションのUIには多数の動的機能が使用されていて、動的UIを理解するための出発点として最適です。

ウィジェットの無効化

最も単純な形の動的UIは、使用不能なメニュー項目またはボタンを灰色で表示するUIです。UIウィジェットの無効化は、すべてのウィジェットに対して同じように作用します。setEnabled()関数は、Componentクラスの機能です。リスト1に、ボタンを無効化するコードを示します。

リスト1. ボタンの無効化

```
button.setEnabled(false);
```

使用できないメニュー・オプションまたはダイアログ・ボックスのボタンを灰色表示にするという単純なアクションでも、ユーザーにとってトレードオフがあります。ボタンが灰色表示になっていれば、ユーザーは特定のアクションを実行できないことがすぐにわかりますが、実行できない理由はわかりません。理由を理解できないユーザーにとって、これが問題になるかもしれません（「一般原則」を参照）。

見てのとおり、非常に簡単です。問題は、いつボタンを有効または無効にするかです。一般的な設計上の決定は、適用不能なときにボタンを無効にするというものです。例えば、多くのプログラムは、ファイルが最後に保存されてから変更されていないときには、Save（保存）ボタン（および対応するメニュー項目）を無効にします。

ボタンを無効にするときの主な注意点は、適切なときに再び有効にするのを忘れないことです。例えば、ボタンのクリックとそのアクションの完了との間に確認ステップがある場合、確認が失敗した場合でも、ボタンを再び有効にしなければなりません。

範囲の調整

ときには、アプリケーションはSpinnerやSliderなどの数値ウィジェットの範囲を動的に調整する必要があります。これは、見かけよりはるかに複雑な場合があります。特にSliderには、目盛り、目盛り間隔、ラベルなど、二次機能があります。表示が醜くなるのを避けるために、範囲とともに調整する必要があるかもしれません。

SwingSet2デモでは、このような調整を直接行わないので、このデモに手を加えて、別のスライダーを変更できるChangeListenerを、スライダーにアタッチします。リスト2に示した新しいSliderChangeListenerクラスを入力します。

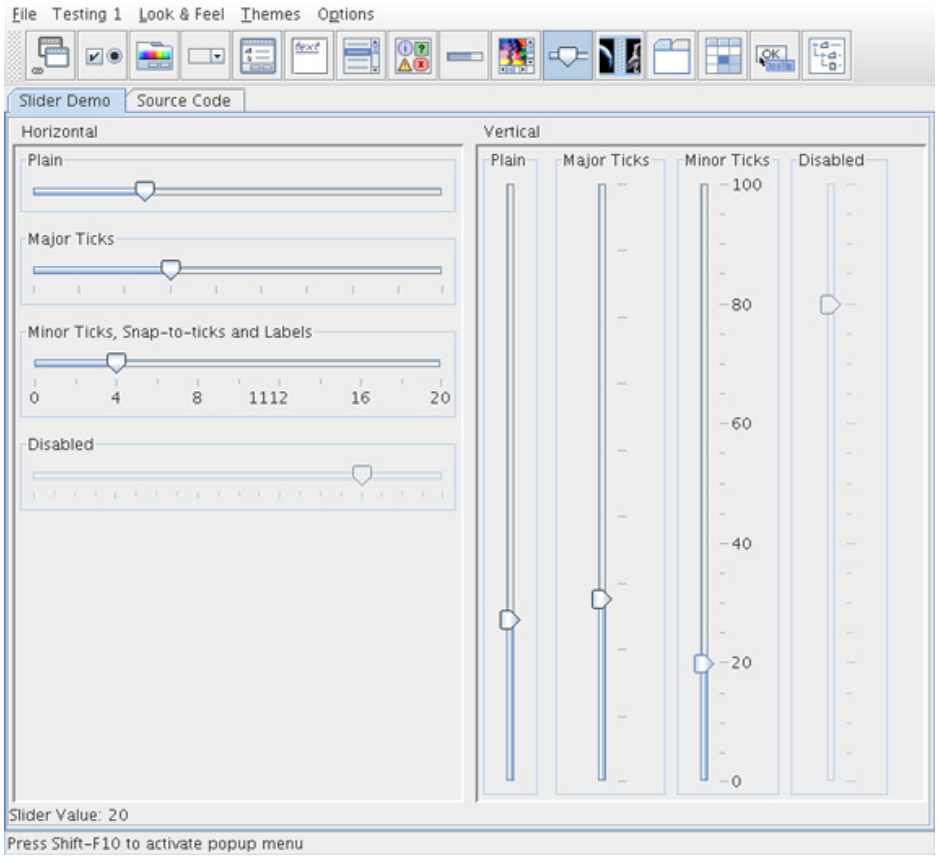
リスト2. スライダーの範囲の変更

```
class SliderChangeListener implements ChangeListener {
       JSlider h;

       SliderChangeListener(JSlider h) {
              this.h = h;
       }

       public void stateChanged(ChangeEvent e) {
           JSlider js = (JSlider) e.getSource();
           int i = js.getValue();

           h.setMaximum(i);
           h.repaint();
       }
}
```

3番目の水平スライダーが作成されると（オリジナル・デモでは、すべての単位に目盛りがあり、5、10、11のラベルがあるスライダー）、新しいSliderChangeListenerも作成されて、スライダーをコンストラクター引数として渡します。3番目の垂直スライダー（範囲が0から100までのもの）が作成されると、新しいSliderChangeListenerが変更リスナーとして追加されます。これは期待通りに機能します。すなわち、垂直スライダーを調整すると、水平スライダーの範囲が変わります。

あいにく、目盛りとラベルはまったく機能しません。5目盛りごとのラベルは、範囲があまり大きくなければ問題ありませんが、図1に示すように、11という余分なラベルがユーザビリティー問題になります。

図1. ラベルの混在



目盛りとラベルの更新

明らかな解決策は、リスト3に示すように、最大値が更新されたら、単に水平スライダーの目盛り間隔を設定することです。

リスト3. 目盛り間隔の設定

```
// DOES NOT WORK
int tickMajor, tickMinor;
tickMajor = (i > 5) ? (i / 5) : 1;
tickMinor = (tickMajor > 2) ?  (tickMajor / 2) : tickMajor;
h.setMajorTickSpacing(tickMajor);
h.setMinorTickSpacing(tickMinor);
h.repaint();
```

リスト3は現状では正しいですが、画面に表示されるラベルは変更されません。setLabelTable()を使用して、ラベルを個別に設定する必要があります。もう1行追加すると、修正できます。

```
h.setLabelTable(h.createStandardLabels(tickMajor));
```

これでも、最初にセットアップされた奇妙なラベル11が残ります。もちろん、目的は、スライダーの右端にラベルを表示することです。このためには、（新しい最大値を設定する前に）奇妙なラベルを除去してから、新しいラベルを追加します。次のコードは、ほぼ問題なく機能します。

リスト4. ラベルの置換

```
public void stateChanged(ChangeEvent e) {
       JSlider js = (JSlider) e.getSource();
       int i = js.getValue();

       // clear old label for top value
       h.getLabelTable().remove(h.getMaximum());

       h.setMaximum(i);

       int tickMajor, tickMinor;
       tickMajor = (i > 5) ? (i / 5) : 1;
       tickMinor = (tickMajor > 2) ? (tickMajor / 2) : tickMajor;
       h.setMajorTickSpacing(tickMajor);
       h.setMinorTickSpacing(tickMinor);
       h.setLabelTable(h.createStandardLabels(tickMajor));
       h.getLabelTable().put(new Integer(i),
       new JLabel(new Integer(i).toString(), JLabel.CENTER));
       h.repaint();
}
```

一度言ったとすれば、二度言った

「ほぼ」と言ったのは、リスト4のコードでは11のラベルが除去されますが、iに新しいラベルが追加されないためです。代わりに、tickMajor間隔でラベルが表示されるだけです。解決策は、一見、かなり衝撃的なものに見えるでしょう。

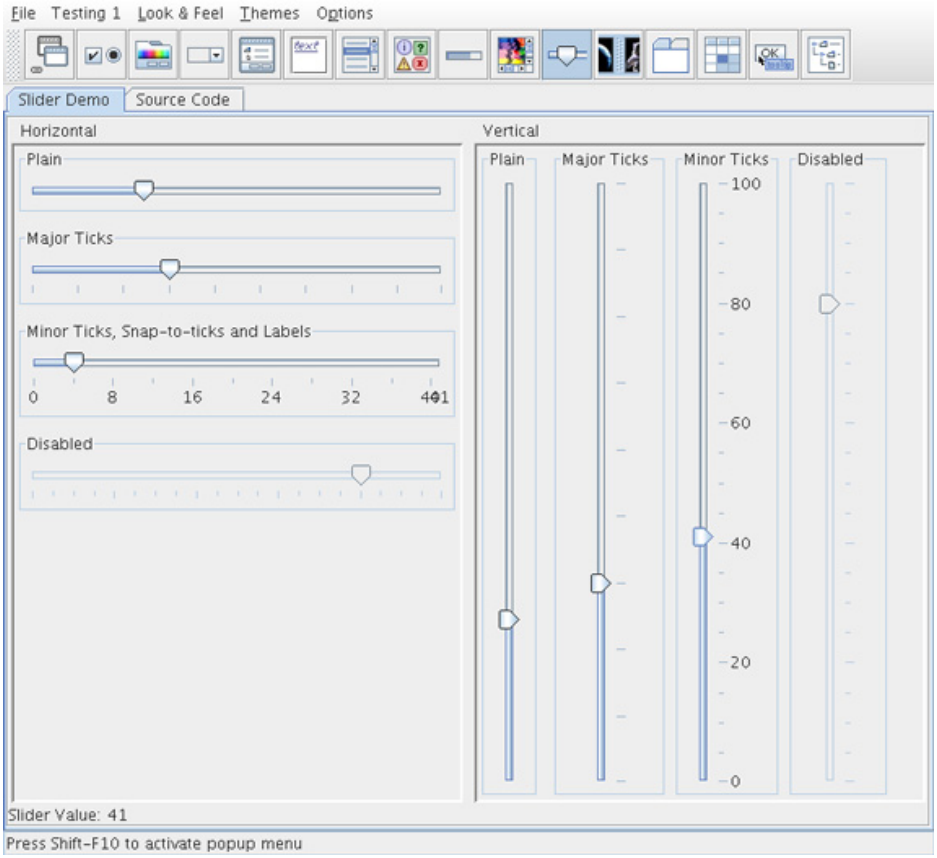
リスト5. 表示の強制更新

```
h.setLabelTable(h.getLabelTable());
```

この一見つかみどころのない操作が、実際には重大な効果を持っています。ラベル・テーブルが設定されると、スライダーのラベルが生成されます。テーブルの変更に対する特別なコールバックはないので、テーブルに新しい値を追加しても、必ずしも効果があるわけではありません。リスト5の明らかなno-opには、表示を更新しなければならないことをSwingに知らせるという副作用があります。（私がこれを発明したわけではありません。オリジナルのSwingSetコードにそのようなコールが含まれているのです。）

これで、残る問題は1つだけです。スライダーの端にラベルを表示したいという目的により、図2に示すように、2つのラベルが隣り合って表示されたり、重なり合って表示されたりすることがあります。

図2. スライダーの端のラベルの重なり



この問題には、数多くの解決策があります。その1つは、自分でコードを書き、ラベル・テーブルに値を入れ、早い段階でシーケンスを中止して、シーケンスの最後のラベルをスライダーの端から離すことです。これは読者の皆さんへの宿題にしましょう。

メニューの更新

多くの場合、メニューの変更をメニュー項目の有効化と無効化に制限することは、きわめて実際的です。この手法は、項目の無効化に適用される一般的な注意事項の対象となります。重要な項目を無効化することによって、プログラムがうっかり使用不能な状態のままになるのを避けてください。

メニュー項目またはサブメニューの追加や削除も可能です。JMenuBarを変更するのは、それほど簡単ではありません。バーから個々のメニューを削除または置換するためのインターフェースがないからです。（バーの右端に新しいメニューを追加する以外で）バーを変更したい場合は、新しいバーを作成して、元のメニューと置き換える必要があります。

個々のメニューの変更は、ただちに有効になります。バーや別のメニューにメニューをアタッチする前にメニューを構築する必要はありません。メニュー・オプションの選択を変更する必要があるとき、最も簡単な方法は、特定のメニューを変更することです。それでもやはり、メニュー全体を追加したり削除したりする場合があるかもしれませんが、これは特に難しくありません。リスト6では、メニュー・バーの、特定のインデックスの前にメニューを挿入する方法の単純な例を示しています。この例では、置き換えられるJMenuBarはJFrameオブジェクトにアタッチされる

ことを前提としていますが、メニュー・バーを取得および設定できるものであれば、どんなものでも同じように機能します。

リスト6. メニュー・バーへのメニューの挿入

```
public void insertMenu(JFrame frame, JMenu menu, int index) {
       JMenuBar newBar = new JMenuBar();
       JMenuBar oldBar = frame.getJMenuBar();
       MenuElement[] oldMenus = oldBar.getSubElements();
       int count = oldBar.getMenuCount();
       int i;

       for (i = 0; i < count; ++i) {
              if (i == index)
                     newBar.add(menu);
              newBar.add((JMenu) oldMenus[i]);
       }
       frame.setJMenuBar(newBar);
}
```

このコードは、私が最初に試みたものではなく、うまく機能するように手を加えた最終バージョンです。いくつか興味深い奇癖を反映しています。一見して、これを実装するにはgetComponentAtIndex()を使うのが当然のように見えるかもしれませんが、これは非推奨になりました。さいわい、getSubElements()インターフェースが十分に役立ちます。newBar.add()の場合、JMenuへの型変換はおそらく安全ですが、私は好きではありません。getSubElements()インターフェースはメニュー・バーだけでなく、メニューにも作用します。メニューにいくつかの型のサブ要素を持たせることができますが、JMenuBarに追加できる要素はJMenuだけです。したがって、JMenuBar.add()メソッドに渡すためには、要素をJMenuに型変換しなければなりません。あいにく、将来のAPIの改訂でJMenu以外の型の要素をJMenuBarに追加できるようになった場合、返された要素をJMenuに型変換するのは不要になり、安全でなくなります。

リスト6のコードでは、もう1つのかなり微妙なインターフェースの奇癖を反映しています。すなわち、あらかじめ、メニュー・カウントをキャッシュしなければなりません。メニューが新しいバーに追加されると、古いメニュー・バーから削除されます。リスト7のコードも同じに見えますが、機能しません。ループが早い段階で終了します。

リスト7. 終了が早すぎるループ

```
// DOES NOT WORK
for (i = 0; i < oldBar.getMenuCount(); ++i) {
       if (i == index)
              newBar.add(menu);
       newBar.add((JMenu) oldMenus[i]);
}
```

> ユーザーはインターフェースの一貫性から恩恵を受けます。特定のメニューは常に同じ場所にあるべきです。ユーザーにとって便利なように、変更の可能性があるメニューはメニュー・リストの右端に置き、変更されないメニューは左側の固定位置に置くようにしましょう。同様に、可能なら、項目は常にメニュー内の同じ位置に置いてください。メニュー項目の位置がころころ変わるよりは、灰色表示の方が、ユーザーにとって目障りではありません。メニューの他の項目の位置が変わらないからです。

リスト7のループは、半分の項目しかコピーしません。例えば、メニュー・バーにそもそも4つの項目があった場合、はじめの2つしかコピーされません。最初の項目をコピーすると、iは1となり、getMenuCount()は3を返します。2番目の項目をコピーすると、iは2となり、getMenuCount()

は2を返すので、ループは終了します。バーにメニューを追加すると、別のバーから削除されるという「機能」について記載された資料を見つけることができなかったため、意図的なものではないのかもしれません。それでも、これの対処法は簡単です。

メニュー・バーからメニューを削除するのは、もう少し簡単です。元のバーの他のメニューをすべて、新しいバーにコピーすればよいのです。簡単ですね！

インターフェースでメニューの動的更新を多用する場合は、メニューを常にその場で更新するよりも、メニュー・バーのセットを作成して、セットを切り替えた方がよいかもしれません。ただし、あまり頻繁にメニューを変更すると、ユーザーは混乱するでしょう。

ウィンドウのサイズ変更

うれしいことに、通常、ウィンドウのサイズ変更は自動的に行われます。しかし、サイズ変更が持つ意味を考えておく必要があります。非常に小さいウィンドウでは、ボタン・バー、メニュー・バー、および同様の機能が問題になることがあります。プログラムが自分で管理するグラフィカル・パネルは、サイズ変更イベントに対応する必要があります。UI要素のパッキングはSwingに任せてもよいですが、コンポーネントのサイズに気を付けていてください。一度取得した寸法を不用意に使用し続けないでください。

さらに油断できないこととして、スライダーの目盛り間隔など、設計上の決定のなかには、ウィンドウのサイズ変更イベントに応じて適度に更新されるものがあります。幅100ピクセルのスライダーには、幅400ピクセルのスライダーと同じ数のラベルを見やすく表示することはできません。より大きな表示には、まったく新しい利便性機能を追加することによって、UIをさらに充実させた方がよいかもしれません。

とはいえ、ほとんどの場合、ウィンドウ・サイズの変更は無視することができます。ただし、サイズ変更を不必要に妨げたり、オーバーライドしたりしてはいけません。レイアウト・コードには、取るに足りない利便性は必要ありません。最小ウィンドウ・サイズは調整可能かもしれませんが、ユーザーが好きなだけウィンドウを大きくできるようにしておきましょう。

一般原則

Swingツールキットは、UI設計に大幅な柔軟性を与えてくれます。注意して使用すれば、インターフェースをその場で更新するオプションにより、インターフェースは劇的に単純化できます。例えば、メニュー・オプションが適用されるときだけメニューを表示すれば、ユーザーにとって使い勝手がよくなります。

残念ながら、この手法を可能にするAPI機能の中には、やや気まぐれなものがあり、副作用と対話が常に詳しく解説されているわけではありません。動的インターフェースについてのアイデアが浮かんだら、デバッグに少しだけ余分に時間を費やす心積もりをしてください。Swingライブラリーの隅々を調べると、驚くべき動作やバグに対処する必要があることがわかるかもしれません。

明白な実装がなくても、がっかりしないでください。この記事のJMenuBarの例が示すように、あるタスクのサポートがAPIになくても、自分で実装することができるかもしれません（やや間接的ではあっても）。

極端に走るのはやめましょう。動的UIの真価は、本質的な制限をユーザーにわかりやすくすることにあります。理想的には、ユーザーが、インターフェースが変化したことに気づかないことです。選択されたオブジェクトがあるときだけ、プログラムのObjectメニューが使用できるようになっていれば、その他の時間にObjectメニューがなくても、ユーザーは気にしません。

一方、オプションが使用できない理由をユーザーが推測できない可能性がある場合は、ユーザーにアクションを実行させて、わかりやすいエラー・メッセージを表示した方がよいかもしれません。これは特に一部のアクションで重要です。保存オプションが無効になっているだけでは、データを保存したい場合にユーザーは不思議に思います。おそらく、プログラムはデータがすでに保存されていることを認識していますが、ユーザーは、それに気づいてなく、保存できないことを疑問に思います。また、ファイルを保存できない特定の理由があるのであれば、それを知りたいと思うでしょう。

インターフェース設計は、長年の研究にもかかわらず、まだまだ多くの点で未熟な分野です。少し実験してみてください。UIの動的変更は、UIをわかりやすく、シンプルにし、応答性を高める素晴らしい機能です。動的UI機能を追加するには、数分間の作業からかなり長時間の作業まで、ある程度の労力が必要です。

著者について

Peter Seebach

Peter Seebachは、たしかこの辺にFileメニューがあったはずだと思っています。Peterはコンピューターを使い始めてすぐに、プログラミングを始めましたが、いまだにGUIを目新しく感じています。