Swing コードをデバッグし、テストする

他の人が作成した Swing コードを理解するためのツールと手法

Alex Ruiz
Consulting Member of Technical Staff
Oracle

2010年 2月 02日

他の Java™ 開発者のコードを使用したり保守したりする必要がある場合、デバッグやテストをすると、そのコードの動作を理解しやすくなります。しかし GUI のコードの場合には、それに合った適切なツールがない限り、効果的なデバッグやテストを実践するのは困難です。この記事で紹介する 2 つのオープンソース・ツール、Swing Explorer と FEST-Swing を使用すると、Swing UI のデバッグやテストが簡単に行える上に信頼性の高いものになります。この記事では、著者の Alex Ruiz がこの 2 つのツールを使って UI の構造を理解する方法、UI の動作をテストする方法、そして問題のトラブルシューティングを行う方法について説明します。

Swing は今日利用可能な GUI ツールキットの中でも強力なものの 1 つです。Swing は拡張や構成が容易な、クロスプラットフォームのツールキットです。しかし Swing の柔軟性は大きな強みであると同時に大きな弱みでもあります。Swing を使用する場合、同じ UI を異なる方法で作成することができます。例えば、GUI コンポーネント間の間隔を空けるために、インセットや、空のボーダー、そして (指定されたサイズの形状を生成できる) フィラーを使うこともできます。Swing には膨大なオプションがあるため、既存の GUI を理解することは新しい GUI を作成することと同じくらい大変な作業であり、また GUI として表示されるコンポーネントとそのコードとを対応させることは簡単ではありません (`GridBagLayout` を使用する数行のコードを読みながら GUI を頭に描いてみてください)。

他の人が作成した Swing GUI を保守する場合であれ、サードパーティーの GUI コンポーネントをアプリケーションに統合する場合であれ、そのコードを理解するための適切な手段は、テストを作成してみることです。テストを作成しながら、未知のコードの内部を探っていくのです。そうした作業による貴重な副産物として、コードの保守によって他の部分に悪影響が及ぶのを防ぐ上で役立つテスト・スイートを得ることができます。サードパーティーの GUI コンポーネントを統合する場合には、ライブラリーの新しいバージョンによって何か動作が変更されていないかどうかを、そうしたテスト・スイートによって見つけることができます。

適切な出発点としては、ユーザー入力に対して GUI がどのように動作するかを理解するための機能テストを作成することです。GUI 用のテストの作成は、GUI コンポーネント以外のコード用のテストの作成よりも複雑です。その理由は以下に挙げるとおりです。

- 理想的にはテストを自動化する必要がありますが、GUI は人間が使う前提で設計されており、コンピューター・プログラムが使う前提では設計されていません。
- 個々のクラスを個別にテストする従来のユニット・テストは GUI コンポーネントのテストには適していません。GUI で言う「ユニット」では、複数のクラスから構成される複数の GUI コンポーネントが連携して動作します。
- GUI はユーザーによって生成されたイベントに応答します。GUI をテストするためには、ユーザー入力をシミュレートし、生成されたイベントがすべてのリスナーにブロードキャストされるまで待ち、その結果が GUI としてユーザーに表示されるものと同じかどうかをチェックする、といった手段が必要です。ユーザーと GUI とのやり取りをシミュレートするコードの作成は面倒であり、間違いを起こしがちです。
- GUI のレイアウトを変更しても機能テストには影響しない堅牢さが必要です。

もう 1 つの問題として、テスト対象の GUI の構造と動作をあらかじめ知っておく必要があります。そうでないと、どのコンポーネントを自動テストで使用するのか、また何を検証すればよいのかを判断することができません。一般に、GUI のテストを作成するためには以下の情報が必要になります。

- GUI の中にあるテスト対象のコンポーネントはどれか
- そうしたコンポーネントをテストの中で一意に識別する方法
- ある特定の使用状況で想定される、コンポーネントの状態 (またはプロパティー)

ビジュアル設計ツール (NetBeans Matisse など) を使用することで GUI の構造を理解できる場合もあります。しかしそうした種類のツールは設計時の GUI の情報しか表示することができません。設計時の情報と実行時に表示される情報とは異なる可能性があります。例えば、一部のコンポーネントはユーザー入力に応じて表示と非表示が切り換えられるかもしれません。

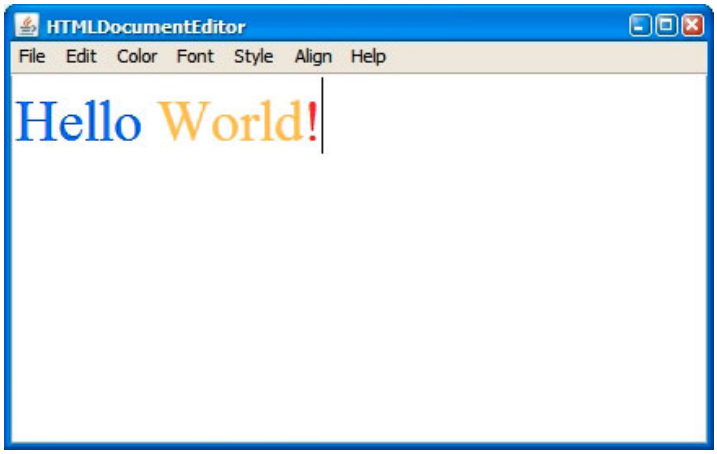
従来のデバッガーでは、ある使い方で実行されている GUI の状態を知ることはできません。Swing コードの中に設定されたブレークポイントでデバッガーが停止すると、GUI による描画が中断され、GUI のところには空白の四角が表示されてしまいます。理想的には、デバッガーで GUI をステップ実行させながら GUI の動作を見られる必要があります。

幸いなことに、Swing Explorer と FEST-Swing という 2 つのオープンソース・ツールを利用すると、既存の Swing コードを理解する作業が必要な場合に、その作業を迅速に進めることができます。この記事では、この 2 つのツールを紹介しながら、この 2 つを組み合わせてアプリケーションの GUI 構造を検証する方法、GUI の機能をテストする方法、そして潜在的な問題を特定する方法を説明します。

検討対象のアプリケーション

この記事で説明する例の大部分では、テスト対象のアプリケーションとして、HTML を編集するための基本的な機能を持つ、HTMLDocumentEditor という無料の HTML エディターを使用します (「参考文献」を参照)。これらの例を皆さんが自分で試したい場合には、このアプリケーションとサンプルのテスト・コードをダウンロードしてください。図 1 は HTMLDocumentEditor が実際に動作している様子を示しています。

図 1. HTML エディター



GUI のテストを作成する前に、GUI の構成を理解する必要があります。この HTML エディターは非常に単純であり、テキスト領域といくつかのメニューで構成されています (これらのメニューによって HTML 文書を開き、保存し、編集します)。

また、各コンポーネントが具体的にどんなタイプなのかを理解することも重要です。それによって、GUI コンポーネントがどのような動作をし、どのような性質をもつのか、またそれらをどのように API を通じてテストに使用すればよいのかを理解しやすくなります。この HTML エディターの場合には、テキスト領域が `JTextArea` なのか`JTextPane` なのか、あるいはカスタム GUI コンポーネントなのかを判断する必要があります。GUI コンポーネントのタイプを判断するための 1 つの方法は、ソース・コードを検証することです。GUI がどのように実装されているかによって、ソース・コードの検証は容易な作業にも、困難な作業にもなりえます。HTMLDocumentEditor のソース・コードは読みやすく、容易に内容を理解することができます。この場合には、ソース・コードを少し調べてみると、テキスト領域が `JTextPane` であることがわかりますが、おそらく皆さんは仕事を進める中で、あまりうまく作成されていないため理解しにくい GUI コードに突き当たることがあるはずです。そうした場合には、そのコードを解読するために大量の時間を費やすか、あるいは効率的に解読してくれるツールを見つける必要があります。

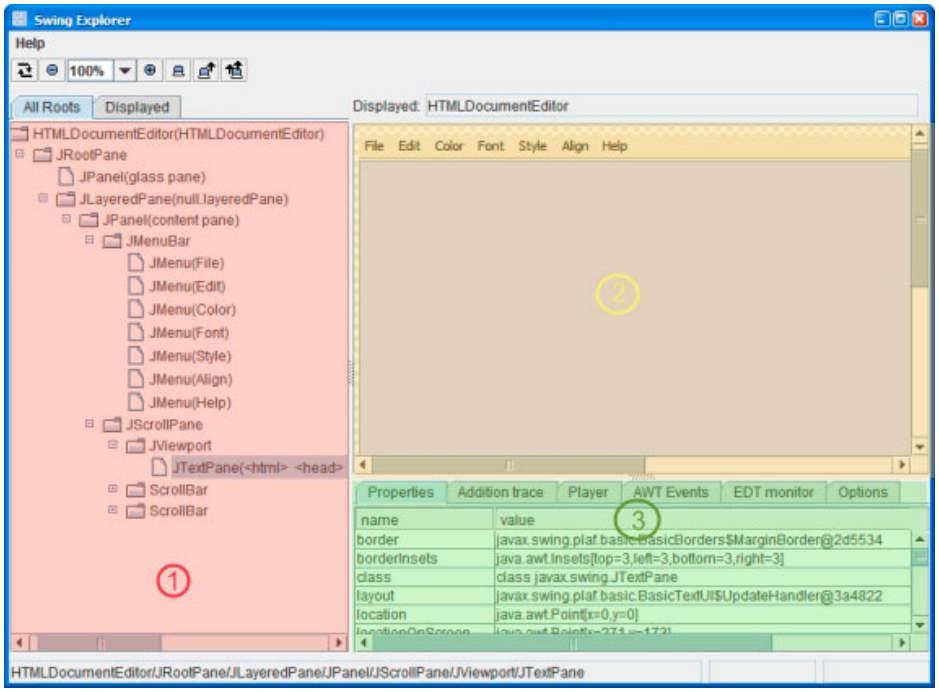
Swing Explorer の紹介

Swing Explorer を利用すると、Swing GUI の内部を視覚的に検証することができます (「参考文献」を参照)。Swing Explorer の UI は単純で直感的であるため、GUI のコンポーネントの検出、コンポーネントの描画方法、コンポーネントのプロパティーの検証、等々を、いつでも容易に行うことができます。

Swing Explorer は、スタンドアロンのアプリケーションとしても、Eclipse や NetBeans 用のプラグインとしても配布されています。Swing Explorer を使う推奨の方法は、IDE のプラグインとして使用する方法です。この記事では Eclipse のプラグインとして Swing Explorer を使用します (「参考文献」を参照)。

Swing Explorer プラグインをインストールしたら、Swing Explorer を使用して、この HTML エディターのメイン・クラスを起動します (図 2)。

図 2. Swing Explorer の中で起動されたエディター・アプリケーション

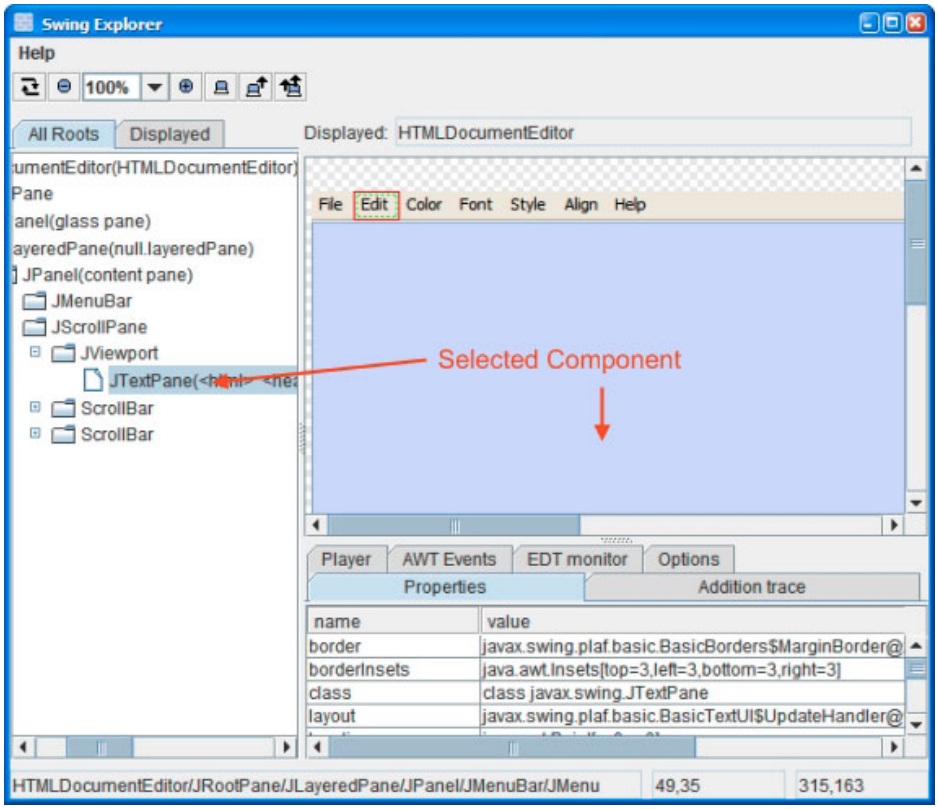


Swing Explorer には以下に挙げるビューがあり、これらのビューは Swing GUI の内部を理解する上で役に立ちます。

1. コンポーネントの階層構造を表示するツリー・ビュー
2. 検証対象の GUI
3. 選択されたコンポーネントのプロパティー (名前、サイズなど) を表示し、また他の有用で興味深いツール (「参考文献」を参照) を含むタブ付きペイン

Swing Explorer を使用すると、GUI の構造を容易に理解することができます。この演習では、HTML エディターのテキスト領域として使用されているコンポーネントのタイプが、ソース・コードを見ても判断できないものとします。Swing Explorer を使用すれば、単純にそのテキスト領域をコンポーネント・ツリー・ビューで選択するか、あるいは表示された GUI の中でそのコンポーネントをクリックするだけです。下記の図 3 を見ると、そのテキスト領域が `JTextPane` であることを Swing Explorer によって確認できることがわかります。

図 3. 選択されたコンポーネントのプロパティーを Swing Explorer で表示する



アプリケーションの動作を理解し、テストする

テスト対象の GUI の構造が明らかになったら、次のステップはアプリケーションの動作を理解することです。動作を理解できると、どのような想定事項を検証すればよいのかがわかります。アプリケーションの動作を理解する方法としては、現在のエンド・ユーザーに尋ねてみる方法や、アプリケーションのドキュメントを読む方法 (ドキュメントがある場合)、またはそのアプリケーションを皆さん自身が使用してみる方法もあります。

ここでは出発点として、下記の 2 つのテスト・ケースを行うことにします。

1. HTML ファイルを開く
2. 文書のフォントの色を変更する

これで、GUI の機能テストを作成する準備が整いました。

GUI の機能テストによって、アプリケーションが想定どおりに動作することを検証します。このテストでは、GUI の外観ではなくアプリケーションの動作に焦点を絞ります。GUI の堅牢な機能テストを作成するためには、以下の要素が不可欠です。

- ユーザー入力 (キーボードとマウス) をシミュレートできること
- GUI コンポーネントを検索するための信頼性の高いメカニズムがあること
- コンポーネントの位置やレイアウトの変更を許容できること

不適切な方法: `Robot` を直接使用する

自動テストによって確実にユーザー入力をシミュレートするためには、あたかもユーザーがキーボードとマウスを使用しているかのように、オペレーティング・システム・レベルで「ネイティブ」イベントを生成する必要があります。JDK はバージョン 1.3 以来、AWT (Abstract Window Toolkit) の `Robot` により入力シミュレーションの機能を提供しています。しかし `Robot` は Swing コンポーネントを参照して動作するわけではなく、画面の座標を利用して動作するため、直接 `Robot` を使用したテストは脆弱なものになってしまいます。つまり、少しでもレイアウトを変更するとテストが動作しなくなります。

また AWT の `Robot` は下位レベルでの動作にしか対応していないため、マウス・ボタンのクリックやキーボード入力をシミュレートすることしかできません。「このコンボ・ボックスの 3 番目の要素を選択する」などの上位レベルのアクションをシミュレートする場合には、上位レベルのアクションを `Robot` アクションのレベルまで落とし込むためのコードを作成する必要があります。この作業は、テストで必要なアクションの数や、関係するコンポーネントのタイプ次第では、膨大なものになりかねません。しかも AWT の `Robot` には、コンポーネントを検索する (「OK」というラベルが付けられたボタンを見つける、など) ための信頼性の高いメカニズムがありません。そうした場合にも、独自のコードを作成する必要があります。

一般に、AWT の `Robot` を直接使用するには、かなりの労力と時間が必要です。しかし、GUI の機能テストを作成する場合には、GUI のテストを行う上でベースとなる部分ではなく、検証対象の動作に焦点を絞る必要があります。

FEST-Swing の紹介

FEST (Fixtures for Easy Software Testing) Swing モジュールは、GUI の堅牢な機能テストの作成および保守を容易にするためのライブラリーです。FEST-Swing の主な特徴を以下に挙げます。

- AWT の `Robot` をベースに構築されており、実際のユーザー入力をシミュレートすることができます。
- 直感的で理解しやすく、コンパクトで柔軟なインターフェースによって、GUI の機能テストの作成や保守が容易になります。リスト 1 は上位レベルのアクションをコーディングする方法を示しています。このコードは「`firstName`」というテキスト・フィールドに「luke」というテキストを入力し、「ok」ボタンをクリックする動作を記述しています。

  ### リスト 1. FEST-Swing の柔軟なインターフェース

  ```
  dialog.textBox("firstName").enterText("Luke");
  dialog.button("ok").click();
  ```

- GUI コンポーネントの状態を検証するためのアサーション・メソッドがあります。リスト 2 には、「answer」という名前のラベルのテキストが「21」であることを検証するアサーションを示しています。

  ### リスト 2. FEST-Swing のアサーション

  ```
  dialog.label("answer").requireText("21");
  ```

- レイアウト変更で動作不能となることのない、堅牢なテストを作成することができます。
- JDK に含まれている Swing コンポーネントをサポートしています。

- JUnit 4 と TestNG をサポートしています。
- Swing スレッドが適切に使用されているかどうかを検証することができます。
- 失敗したテストのトラブルシューティングが容易になります。

FEST-Swing を使用して GUI の機能テストを作成する

ここまでのところで、HTML エディター・アプリケーションの GUI の構造を理解し、実際のテスト・ケースを確定し、信頼性の高いテスト・ツールを見つけたので、これで GUI の機能テストを作成する準備が整いました。

テスト・ケース: HTML ファイルを開く

HTML エディターでファイルを開くためには、以下を行う必要があります。

1. File > Open の順にメニューを選択します。
2. 表示されたファイル・チューザーの中で、どのファイルを開くかを選択します。
3. そのファイルの内容をエディターがロードしたかどうかを検証します。

リスト 3 は、このテスト・ケースのコードを示しています。

リスト 3. HTML ファイルを開く操作をテストする

```
public class HTMLDocumentEditor_Test extends FestSwingJUnitTestCase {


   private FrameFixture editor;

   protected void onSetUp() {
     editor = new FrameFixture(robot(), createNewEditor());
     editor.show();
   }

   @RunsInEDT
   private static HTMLDocumentEditor createNewEditor() {
     return execute(new GuiQuery<HTMLDocumentEditor>() {
       protected HTMLDocumentEditor executeInEDT() {
         return new HTMLDocumentEditor();
       }
     });
   }

   @Test
   public void should_open_file() {
     editor.menuItemWithPath("File", "Open").click();
     JFileChooserFixture fileChooser = findFileChooser().using(robot());
     fileChooser.setCurrentDirectory(temporaryFolder())
                .selectFile(new File("helloworld.html"))
                .approve();
     assertThat(editor.textBox("document").text()).contains("Hello");
   }
}
```

リスト 3 で行われているテストの詳細は以下のとおりです。

- 1 行目は FEST-Swing の `FestSwingJUnitTestCase` を継承しています。FestSwingJUnitTestCase は FEST-Swing の `Robot` を自動的に作成し、Swing のスレッド処理が適切かどうかを検証し (これについては後ほど説明します)、リソースを自動的にクリーンアップします (開いたウィンドウを破棄し、マウスとキーボードを解放する、など)。

- `editor = new FrameFixture(robot(), createNewEditor());` は Frame 上でユーザー入力をシミュレートできる新しい `FrameFixture` を作成し、`Frame` 内部のコンポーネントを (さまざまな検索基準を使用して) 検索し、Frame の状態を確認します。
- `editor.show();` は画面に HTML エディターを表示します。
- `@RunsInEDT` は EDT (Event Dispatch Thread) で `createNewEditor()` メソッドが必ず実行されるように記述してあります。
- `return execute(new GuiQuery<HTMLDocumentEditor>()` は EDT の中に `HTMLDocumentEditor` の新しいインスタンスを作成します。
- `editor.menuItemWithPath("File", "Open").click();` によって、FEST-Swing はユーザーが File > Open の順にメニューをクリックする動作をシミュレートします。
- `JFileChooserFixture fileChooser = findFileChooser().using(robot());` では、FEST-Swing は、HTML エディターによって起動される「Open File」という `JFileChooser` を検索します。
- その次の 3 行で、FEST-Swing は、システムの一時フォルダーの中にある helloworld.html ファイルをユーザーが選択する動作をシミュレートしています。
- `assertThat(editor.textBox("document").text()).contains("Hello");` は、ファイルがエディターにロードされたかどうかを、そのファイルに Hello という単語が含まれるかどうかチェックすることで検証しています。

リスト 3 では `JTextPane` を名前 (`editor`) で検索していることに注意してください。これはテストの中でコンポーネントを検索するための最も信頼性の高い方法です。こうしておけば、たとえ将来 GUI のレイアウトが変更されたとしても、コンポーネントの検索が失敗することはありません。

テスト・ケース: 文書のフォントの色を変更する

HTML エディターによって文書のフォントの色が黄色に変更されるかどうかを検証するためには、以下の操作を行う必要があります。

1. Color > Yellow の順にメニューを選択します。
2. エディターに何かを入力します。
3. 入力されたテキストが黄色であることを検証します。

リスト 4 は、これを FEST-Swing を使って行う方法を示しています。

リスト 4. 文書のフォントの色が変更されるかどうかをテストする

```
@Test
public void should_change_document_color() {
  editor.menuItemWithPath("Color", "Yellow").click();
  JTextComponentFixture textBox = editor.textBox();
  textBox.enterText("Hello");
  assertThat(textBox.text()).contains("<font color=\"#ffff00\">Hello</font>");
}
```

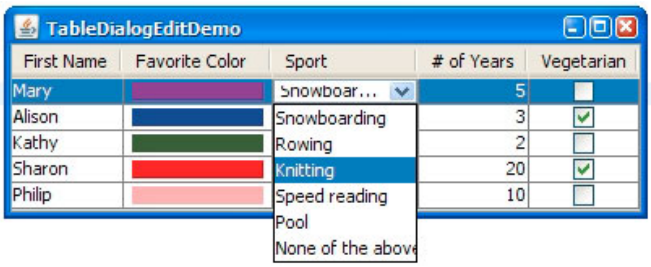
ここまでは、メニューやテキスト・フィールドといった単純な GUI コンポーネントをテストする方法を説明しました。次に、あまり単純ではないテスト・シナリオについて説明します。

より複雑なテスト

FEST-Swing の API がいかに直感的で簡潔かを示すために、ここでは Swing のコンポーネントの中で非常に複雑なものの 1 つである、`JTable` を使うことにします。

ここでは Sun の Swing チュートリアルから引用した `TableDialogEditoDemo` アプリケーションを使用します (「参考文献」を参照)。このアプリケーションは `JComboBox` と `JCheckBox` というカスタム・エディターを持つ `JTable` を使います (図 4)。

図 4. `TableDialogEditDemo`



一例として、ユーザーが「行 0」のコンボ・ボックスの 2 番目の要素を選択する場合をシミュレートするテストを作成します。このテストによって実行されるアクションは以下のとおりです。

1. 必要に応じてテーブルを上下にスクロールし、「行 0」が表示されるようにします。
2. 「行 0、列 2」のセルをクリックします。
3. コンボ・ボックスが表示されるのを待ちます。
4. コンボ・ボックスが表示されたら、そのコンボ・ボックスをクリックします。
5. コンボ・ボックスから、3 番目の要素を選択します。

これはコーディングが必要なアクションの概要を記述したものにすぎません。実際のコードの作成は単純ではありません。幸いなことに、FEST-Swing の API を使用すると、この作業が非常に単純になります (リスト 5)。

リスト 5. 「行 0、列 2」のセルのコンボ・ボックスの 3 番目の要素を選択する

```
dialog.table.enterValue(row(0).column(2), "Knitting");
```

FEST-Swing によって、たとえ複雑な GUI テストであっても、テストが作成しやすく、また理解しやすくなることがわかります。

Swing のスレッド処理

Swing は単一スレッドの UI ツールキットです。Swing はスレッド・セーフではないため、すべての Swing コードを EDT の中で実行しなければなりません。公式のドキュメントに記述されているとおり、複数のスレッドから Swing コードを呼び出すと、スレッドの衝突によるエラーやメモリーに一貫性がないことによるエラーが発生する危険があります (「参考文献」を参照)。

Swing のスレッド・ポリシーには以下のように記述されています。

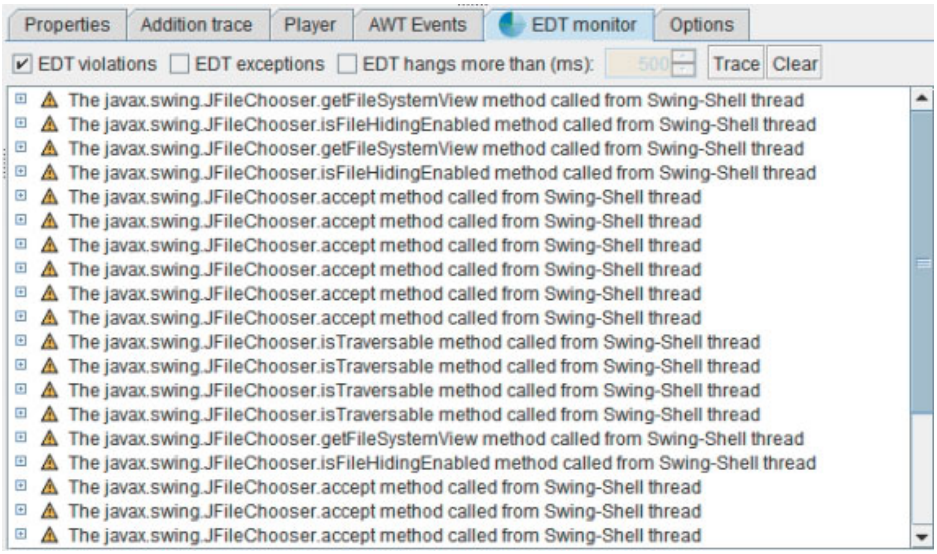
- Swing コンポーネントの作成は EDT の中で行わなければなりません。
- Swing コンポーネントへのアクセスは、スレッド・セーフと記述されているメソッドを呼び出す場合を除き、EDT の中で行わなければなりません。

これらのルールは簡単に守れるように思えますが、極めて簡単に破られてしまいます。Swing には EDT が適切に使われているかどうかを実行時にチェックする機能がなく、ほとんどの場合、

明らかに「適切に振る舞う」と思える Swing UI が、実際にはこうしたルールを破ってしまうのです。

Swing Explorer と FEST-Swing はどちらも、Swing のスレッド・ポリシー違反を検出する機能を持っています。図 5 は Swing Explorer の EDT モニターを示しています。EDT モニターはアプリケーションの実行中に発生した EDT のアクセス違反をレポートすることができます。

図 5. Swing Explorer の EDT モニター



FEST-Swing には、EDT のアクセス違反をチェックし、アクセス違反を検出すると強制的にテストを失敗させる `FailOnThreadViolationRepaintManager` があります。`FailOnThreadViolationRepaintManager` の構成方法は簡単です。セットアップ・メソッドに `FailOnThreadViolationRepaintManager` を配置して `@BeforeClass` というアノテーションを付加すればよいだけです (リスト 6)。

リスト 6. `FailOnThreadViolationRepaintManager` をインストールする

```
@BeforeClass public void setUpOnce() {
  FailOnThreadViolationRepaintManager.install();
}
```

あるいは、UI テストの中で FEST-Swing の `FestSwingTestngTestCase` または `FestSwingJunitTestCase` をサブクラス化する方法もあります (どちらにも `FailOnThreadViolationRepaintManager` が既にインストールされています)。また FEST-Swing には、Swing コンポーネントへのアクセスが必ず EDT の中で行われるように保証する便利な抽象化機能も用意されています。詳しくは「参考文献」を参照してください。

失敗した GUI テストのトラブルシューティング

どのようなライブラリーを使用して GUI の機能テストを作成したとしても、この種のテストは環境関連のイベントの影響を受けがちです。FEST-Swing も例外ではありません。例えば、定期的に実行されるウィルス・スキャンによって、テスト中の GUI に重なるようにダイアログがポップ

アップ表示されるかもしれません。すると FEST-Swing の `Robot` はGUI にアクセスすることができず、結局タイムアウトしてしまい、必然的にテストは失敗します。失敗の原因はプログラミングの誤りではなく、タイミングが悪かったにすぎません。

FEST-Swing には、テストに失敗した瞬間のデスクトップのスクリーン・ショットを取ることができる便利な機能があります。このスクリーン・ショットは自動的に JUnit レポートまたは TestNG のレポートに組み込まれるようにすることも、IDE の中から 1 つのテストだけを実行している場合にはディレクトリーに保存されるようにすることもできます。図 6 は GUI テストに失敗した場合の JUnit による HTML レポートです。このレポートには、テストに失敗した瞬間に撮られたデスクトップのスクリーン・ショットへのリンクが FEST-Swing によって追加されていることに注意してください。

図 6. 失敗したテストからデスクトップのスクリーン・ショットにリンクが張られている、JUnit による HTML レポート

テストが失敗する典型的な原因のもう 1 つが、コンポーネントの検索の失敗です。コンポーネントを検索する上で推奨の方法は、コンポーネントに固有の名前で検索する方法です。しかし場合によると、テスト対象の GUI には各コンポーネントに固有の名前が付いておらず、カスタムの検索基準を使わざるを得ないかもしれません。コンポーネントの検索の失敗には、下記の 2 つのタイプがあります。

1. GUI コンポーネントが見つからない場合。例えば firstName という名前の JTextField を探しているものの、元の開発者が `JTextField` に `firstName` を割り当てるのを忘れたとします。こうした場合、FEST-Swing はスローされた `ComponentLookupException` の中にコンポーネントの階層構造を含めてくれます。そのため開発者は失敗の原因を容易に見つけることができます。この場合では、コンポーネントの階層構造を検証することによって、`JTextField` に適切な名前が指定されていたかどうか、あるいは実際に `JTextField` が GUI に追加されていたかどうかを調べることができます。リスト 7 はコンポーネントの階層構造が `ComponentLookupException` に含まれている様子を示しています。

   #### リスト 7. コンポーネントの階層構造が含まれている `ComponentLookupException`

   ```
   org.fest.swing.exception.ComponentLookupException:
     Unable to find component using matcher
     org.fest.swing.core.NameMatcher[name='ok', requireShowing=false].

   Component hierarchy:
   myapp.MyFrame[name='testFrame', title='Test', enabled=true, showing=true]
     javax.swing.JRootPane[]
       javax.swing.JPanel[name='null.glassPane']
         javax.swing.JLayeredPane[]
           javax.swing.JPanel[name='null.contentPane']
           javax.swing.JTextField[name='name', text='Click Me', enabled=true]
   ```

リスト 7 でコンポーネントの階層構造を見ると、元の開発者が `JTextField` に不適切な名前を指定したことがわかります。名前は現在、`firstName` ではなく `name` となっています。

2. 複数の GUI コンポーネントが見つかった場合。これは指定された検索基準に複数の GUI コンポーネントが一致した場合に起こります。例えば、誤って 2 つの `JTextField` に `firstName` という名前が指定されたような場合です。こうした場合、`firstName` という名前を持つ `JTextField` を探そうとすると、2 つのコンポーネントが同じ名前を持っているため、その検索は (そして結局テストも) 失敗します。この問題を診断しやすくするために、スローされた `ComponentLookupException` には、検索基準に一致するものとして検出されたすべてのコンポーネントが表示されます (リスト 8)。

リスト 8. 検索基準に一致するコンポーネントのリストが含まれる `ComponentLookupException`

```
org.fest.swing.exception.ComponentLookupException:
  Found more than one component using matcher
  org.fest.swing.core.NameMatcher[
    name='firstName', type=javax.swing.JTextField, requireShowing=false].

Found:
javax.swing.JTextField[name='firstName', text='', enabled=true]
javax.swing.JTextField[name='firstName', text='', enabled=true]
```

膨大な数のコンポーネントを持つ GUI を扱う場合はなおさらですが、スローされた `ComponentLookupException` のなかでコンポーネントの階層構造を検証するのが困難な場合があります。こうした場合にも、Swing Explorer は非常に役立ちます。上記で説明したように、コンポーネントの階層構造の中で任意のコンポーネントのプロパティーを選択して検証するためには、そのコンポーネントを直接クリックするだけでよいのです。コンポーネントの階層構造が大規模な場合の解析は、`ComponentLookupException` によるテキスト・ベースの表現を使って行うよりも、Swing Explorer の GUI で行った方が、はるかに容易です。

まとめ

Swing は強力ですが、複雑です。Swing コードを理解しようとすると、Swing コードの作成と同じくらいの困難を伴います。未知の GUI コードを検証するためのテストの作成は、GUI ベースではないコードのテストの場合よりも複雑です。幸いなことに、Swing Explorer と FEST-Swing を利用することで、そうしたプロセスの退屈さや推測作業から解放されます。Swing Explorer を利用すると、アプリケーションを実行させながら GUI の構造を検証することができます。そしてテスト対象の GUI の構造や動作が理解できると、FEST-Swing の簡潔で直感的な API を使用して GUI の機能テストを作成することができます。柔軟な API の他にも、FEST-Swing には Swing のスレッドが適切に使用されているかどうかの検証機能や、失敗した GUI テストのトラブルシューティングの時間を短くするための機能が用意されています。この記事で説明した内容は、この強力な 2 つのツールによって実現できることの表面的な部分にすぎません。

Swing Explorer と FEST-Swing は非常に有用ですが、さらに優れたソリューションは、あかたも開発者が手作業でコーディングしたかのような Java コードでユーザーとのやり取りを記録する記録/再生ツールです。記録/再生ツールを利用すれば、最短の時間でテスト・スイートを作成することができます。既存の GUI を操作すると、ユーザーによって生成されたすべてのイベントはスクリプトの中に記録されます。そのスクリプトを後で再生し、特定のシナリオでのユーザーによる操

作を再現することができます。既存の記録/再生ツールの大きな弱点は、生成されたテストの保守に手間がかかることです。アプリケーションを少しでも変更すると、すべてのテスト・シナリオを再度記録しなければなりません。また、すべてのテスト・シナリオを記録してしまうと、似たようなシナリオをテストする場合には重複してテスト・コードを作成してしまう可能性があります。記録されるスクリプトは長くなる上に、オブジェクト指向言語の機能を持たない独自言語で記述されることが多いものです。反復的な動作をモジュール化しようとすると、非常に手間がかかり、間違えやすく、また通常は新しいプログラミング言語を学ぶ必要があります。

よく使われていて成熟しているオブジェクト指向言語、つまり Java 言語をベースとする記録/再生ツールを使用することで、開発者は機能豊富な IDE のメリットを活かせるようになります。IDE は退屈で間違えやすい作業 (リファクタリングなど) を短時間で終わる簡単な作業に変えることで、生産性の向上と保守コストの低減を可能にします。それがまさに FEST プロジェクト・チームが現在取り組んでいる作業です。つまり FEST プロジェクトでは、FEST-Swing という Java API を使用して、純粋なオブジェクト指向の GUI テストを生成する記録/再生ツールを作ろうとしています。このツールのプレビュー版は 2010年第 2 四半期に入手できる予定です。

ダウンロード

内容	ファイル名	サイズ
Sample GUI tests for this article	jswingtest-code.zip	1.28MB

著者について

Alex Ruiz

Alex Ruiz は、Java 開発、テスト、オブジェクト指向プログラミング、アスペクト指向プログラミング、そして並行性に関する、あらゆる資料を読むことが大好きです。彼の初恋の相手はプログラミングでした。Alex は、Swing や JavaFX のテストやテスト一般を容易にするための革新的な Java ライブラリー、FEST の作成者です。彼は Oracle で開発ツール組織のソフトウェア技術者として働いています。Oracle に入社する前は ThoughtWorks のコンサルタントでした。