

第3章 イベントとイベントハンドラ

【学習内容とねらい】

Windows 上のプログラムは、マウスクリックや特定キーの入力によって、所定の処理が始まることが普通です。この時のマウスクリックなどの動作（アクション）を「イベント」と呼びます。そしてそのイベント発生時の処理内容を記述したプログラムを「イベントハンドラ」と呼びます。「handle（ハンドル）」にはさばく、あるいは処理するという意味がありますが、イベントに対する処理を行うもの「handler（ハンドラ）」という意味から、イベントハンドラと呼ばれるのです。

あるイベント（アクション）に対して、所定の処理（イベントハンドラ）が起動するというプログラミングのスタイルは「イベント駆動型プログラミング」と呼ばれます。実は「イベント駆動型プログラミング」は（プログラミングの歴史からみると）新しいプログラミング・スタイルであり、Windows の普及によって広まったプログラミング・スタイルと言っていいでしょう。

JBuilder では、各コンポーネントに対して、様々なイベントが用意されており、プログラマは適当な「イベント」を選択するだけで良いようになっています。その中でも代表的なものは「ボタン」コンポーネントに対する「クリック」イベントでしょう。「ボタン」はクリック（押される）ために存在しているようなものだからです。本章では、この「ボタンクリック」イベントを典型的なイベントと捉え、ボタンをクリックした時に何らかの処理を行うプログラムを作成します。この“何らかの”処理を記述する部分がイベントハンドラなのですが、イベントハンドラを記述する事が、プログラマが行う主な仕事です。恐らく皆が「プログラムを書く」ということからイメージされる作業が、このイベントハンドラの作成に当たるでしょう。

いよいよここで、自分で（処理内容を）プログラミングする第一歩を踏み出します。

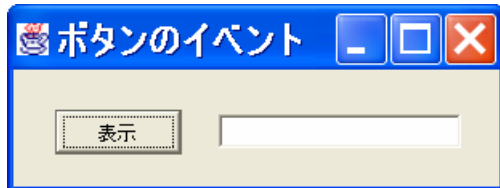
<第3章の構成>

- | |
|----------------------|
| 3-1 ボタンのイベント |
| 3-2 コピープログラムを作ろう |
| 3-3 イベントとイベントハンドラの練習 |

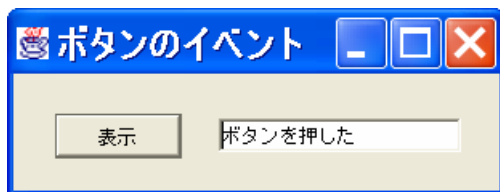
3-1 ボタンのイベント

【基礎課題 3-1-1】

次のようなプログラムを作成してみましょう。動作内容は次の通りです。



プログラムを起動すると、左のような画面が現れる。



[表示] ボタンをクリックすると、テキストフィールド欄に「ボタンを押した」と表示される。

まず、上のようなフレームを持つプログラムを新規アプリケーションとして作ってください。ここに、ボタンとテキストフィールドの `name` プロパティはそれぞれ、「`jButtonDisplay`」、「`jTextField1`」とします。

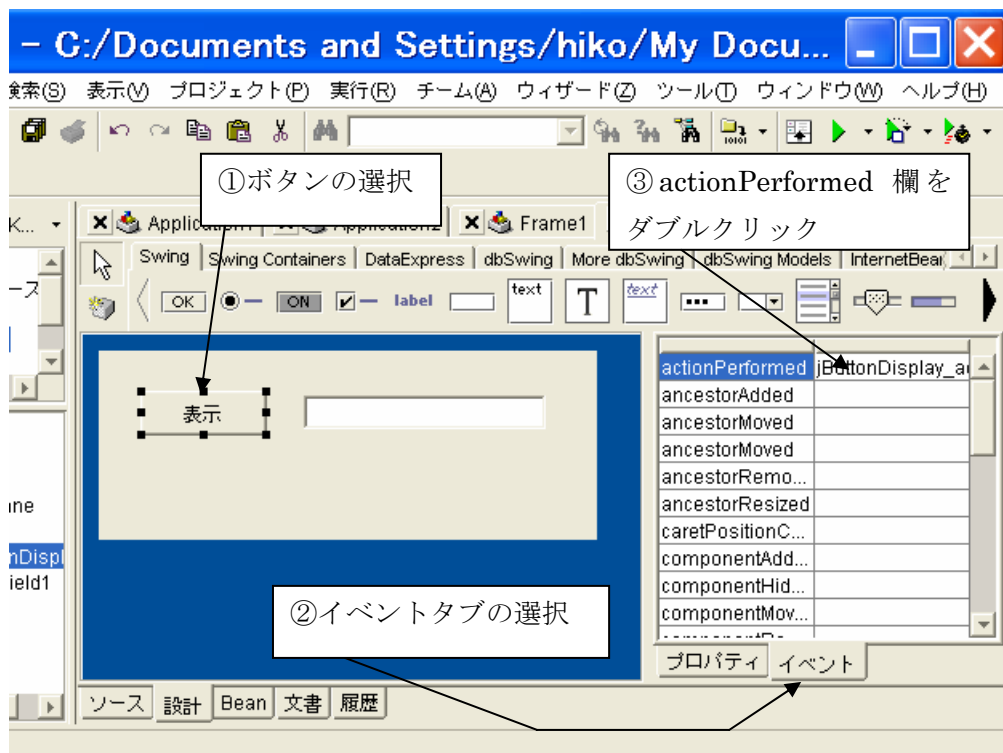
さて、このプログラムの処理は、

1. 「表示」 ボタンをクリックしたときに“ある処理”を行う。
2. その処理とは「テキストフィールドに『ボタンを押した』という文字を表示させる」ことである。

という 2点がポイントになります。

まず、1. から説明します。アプリケーションブラウザの設計ビューから、次の①～③の処理を順に行って下さい（次ページの図参照）。

- ① ボタンコンポーネントを選択する。
- ② インспекタの「イベント」タブを選択する。
- ③ インспекタの最上欄にある「**actionPerformed**」欄をダブルクリックする。



すると、自動的にソースビューに切り替わり、次のプログラムが自動生成されます。

```
void jButtonDisplay_actionPerformed(ActionEvent e) {  
  
}
```

< 解説 >

- ① actionPerformed のアクション (action) とは今の場合、ボタンをクリックする動作 (アクション) を指します。
- ② したがって、actionPerformed はそのアクションが実行された状態、つまりクリックされた状態を意味します。
- ③ 「jButtonDisplay_actionPerfomed」はボタン「jButtonDisplay」がクリックされた時の処理プログラムにつけられた**名前**です。名前ですから何でも良いのですが、どのようなコンポーネントのどのようなアクションに対する処理か、が分かるように、JBuilder が自動的に命名してくれます。
- ④ 具体的な処理内容は **[] で囲まれた部分に記述**します。この課題では、『ボタンを押した』という文字列をテキストフィールドに表示する」処理を書きます。
- ⑤ voidの意味については第6章で学習します。また、()内の「**ActionEvent e**」は**引数**と呼ばれるものですが、現時点では、このように書くものだ、という約束事だと思って下さい (→ActionEventの意味については、6-6 節のコラムで少し解説します)。

それでは、「2. テキストフィールドに文字列を表示する」ことを、Java 言語ではどのように記述するのでしょうか。いよいよ Java 言語プログラミングに入ります。

すでに 2-3 節で学習した通り、「テキストフィールド」コンポーネントの「text」プロパティに入力した文字はテキストフィールドに表示されます。すると、実行時に（手動ではなくプログラムによって）テキストフィールドに文字を表示させるには、

テキストフィールドの text プロパティに文字列を入力

すれば良いことが分かります。Java 言語ではその処理を **setText()** という命令が行います。例えば、`(jTextField)` に文字列「abc」を入力するには次のように記述します。

`jTextField.setText("abc");`

< 解説 >

- ① 「テキストフィールド名. **setText()**」で（ ）内に指定した文字列を text プロパティに入力します。
- ② 文字を" "で囲むと、それは文字列と見なされます。
- ③ 一つの文の終わりには、区切りとして「;」をつけます。
- ④ プログラムの命令は全て半角文字で記述します。
- ⑤ Java 言語では大文字と小文字を区別するので注意してください。

これを用いれば、今求めている処理プログラム（ボタンクリック時に「ボタンを押した」という文字列をテキストフィールドに表示する）を作成することができます。下の空欄を埋めてプログラムを完成させてください。

字下げ ([Tab] キーを押すと、所定の位置に字下げされます。)

```
void jButtonDisplay_actionPerformed(ActionEvent e) {  
    jTextField1.setText(" ");  
}
```

なお、{ }内にプログラムを記述する場合、上のように字下げした方が見やすいプログラムになります。[Tab] キーを用いると良いでしょう。

プログラムを作成したら、実行して動作を確認してみてください……。うまく動作しましたか？ うまく行けば【基礎課題 3-1-1】は終了です。

ここで人間とコンピュータの関係を少し考えてみましょう。

- 「Y」のキーを押すと、画面に「Y」と表示される。
- 「印刷」ボタンを押すと、印刷が始まる。
- アイコンをダブルクリックすると、そのプログラムが起動する。

このように、人間とコンピュータの関係は、

人間が何かをすると、コンピュータがそれに対応して決められた処理をしてくれる。

ようになっています。

人間がコンピュータに対して何かをする時の、“何か（アクション）”を**イベント**といいます。そして、それに対する処理内容を規定したもの（プログラム）を、**イベントハンドラ**といいます。

例えば、上の【基礎課題 3-1-1】の例でいうと、

★イベント : [表示] ボタンのクリック

★イベントハンドラ: 「所定の文字列をテキストフィールドに表示するプログラム」となります。

Windows アプリケーションの場合、「あるイベントが発生すると、それに対応したイベントハンドラが起動する」という形でプログラムが動作します。そして、各コンポーネントには、それぞれ（必要と思われる）「イベント」が用意されています。そこで、プログラム開発者の行う仕事は

- 適当なイベントを（インスペクタの「イベント」タブから）選択する。
- そのイベントに対するイベントハンドラを記述する。

という事になります。

1年生の「情報処理基礎・同演習」では、主に、あるイベントに対してどういうイベントハンドラが起動するのかを学習してきたといえます。

この「プログラミング」では、主にイベントハンドラの作り方と書き方を学習して行きます。

コラム メソッド、オブジェクト、オブジェクト指向プログラミングについて

Java 言語では、`setText()` のような命令を**メソッド**と呼びます。例えば、「`(jTextField.setText())`」の場合は、テキストフィールドに備わっているメソッドと言います。テキストフィールドには、様々なメソッドが備わっていますが、前章でみた通り、**プロパティ**も持っています。このように、プロパティとメソッドを持っている”モノ”を**オブジェクト**と呼びます。コンポーネントは全てオブジェクトです。ちょっと乱暴な説明ですが、このようなオブジェクトを駆使してプログラムを作成する手法を**オブジェクト指向プログラミング**と言います。とりあえず、用語だけでも頭に入れておいて下さい。なお、オブジェクト指向の世界では、一般に「オブジェクト名、メソッド名」の形でオブジェクトに備わっているメソッドを使用できるようになっています。

コラム クラスとオブジェクトについて

ここで、ついでにクラスとオブジェクトの関係について解説しておきましょう。

上のコラムで説明した通り、コンポーネントは全て**オブジェクト**です。そして、(第2章で学習したように) コンポーネントには、テキストフィールドやボタンそしてチェックボックスなど様々な**種類**があります(それらは text プロパティなど共通の性質を持つ反面、それぞれに特徴的なプロパティやさらにはイベントがあります)。これら”種類”あるいは分類の一項目を**クラス**と呼びます。具体的には、Java 言語ではテキストフィールド (**JTextField**) クラスやボタン (**JButton**) クラス、などがあります。

さて、そうすると、クラスとオブジェクトの関係は? という点が問題になりますね。その点を解説しましょう。これまでの経験から、例えばボタンコンポーネントを貼り付けると、「jButton1」、「jButton2」という name が付けられてボタンが生成されました。このように、実際にフレームに貼り付けられた jButton1 などをボタン**オブジェクト**と言います。はたして分かるでしょうか…?

例えて言うなら、クラスとは版画の原盤(版木)のようなものです。これにインクを塗って印刷すると版画作品ができます。できた版画作品がオブジェクトです。原盤さえあれば、オブジェクトである版画作品を生成(印刷)することは容易です。実は、みながボタンコンポーネントを貼り付けた時に **JBuilder** が背後で行っていることは、「原盤である **JButton** クラスを呼び出し、そこから(版画作品のように)オブジェクト jButton1 などを生成する」ことなのです。全てのコンポーネントはそれぞれのクラスから生成されます。そして実はコンポーネント以外にも様々なクラスがあります。

ここで、1-4 節で Java 言語プログラムは「クラスの集まりである」と説明した事を思い出して下さい。実は Java 言語ではクラスの中身を定義すること自体がプログラミングの内容なのです。そこで説明した通り、クラスは次のように定義します。

```
class クラス名 {  
    クラスの定義部分  
}
```

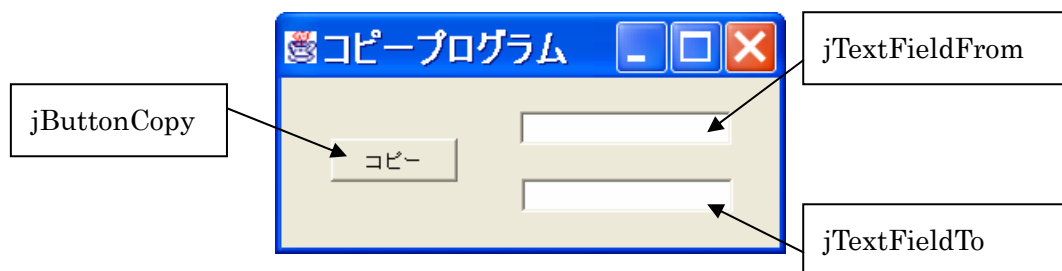
さて、オブジェクトがプロパティやメソッドを持っている事から分かります、上の「クラスの定義部分」は具体的にはプロパティやメソッドの定義を記述する部分なのです。例えば、**JTextField** クラスの定義部分には「**setText()**」メソッドの定義が記述されているという訳です。この、メソッドの定義を自分で行うやり方を第6章で学習します。一方、一般的なクラスを自分で定義する方法については、第7章で簡単に学習します。

とりあえず、以上の点を頭に入れておいて下さい。そうすれば、以降の学習の理解度がより深まるはずです。

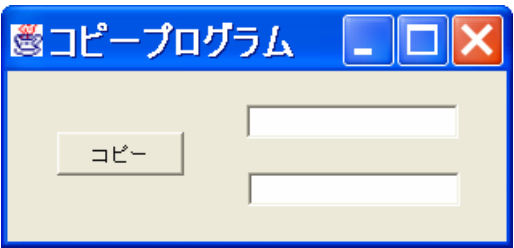
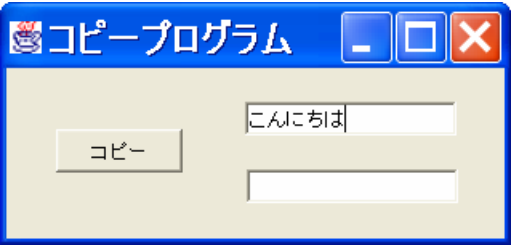
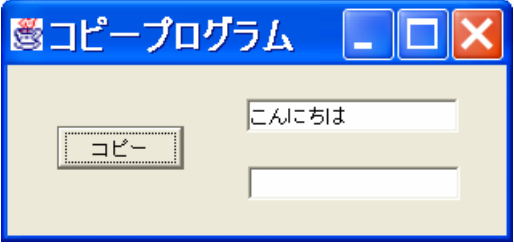
3-2 コピープログラムを作ろう

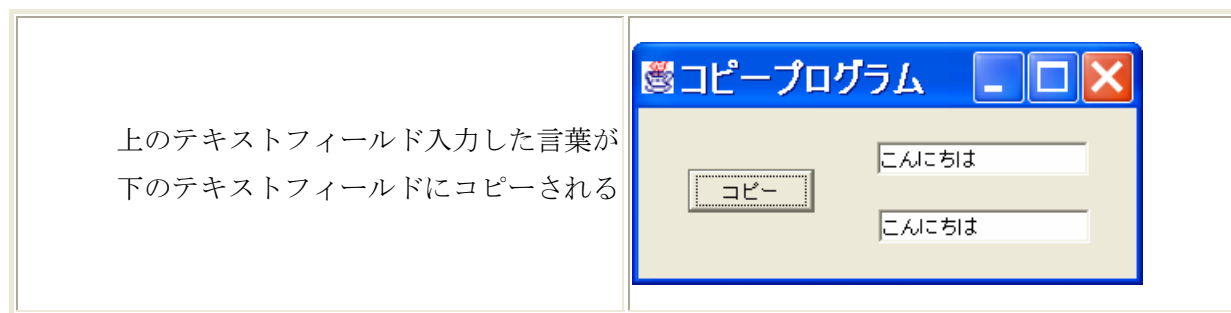
【基礎課題 3-2-1】

ボタンのイベントハンドラを作成する練習として、【基礎課題 2-4-1】でフレーム（のみ）を作成したコピープログラムを改良しましょう。新たに以下のようなフレームを作成してください。各コンポーネントの「name」プロパティは次のように指定します。



コピープログラムとは、

	プログラムを起動して
上のテキストフィールドに言葉を入力して	
	「コピー」ボタンを押すと



というプログラムです。

まず、前節で行った通り、[コピー] ボタンをクリックしたときの（空っぽの）イベントハンドラを生成させてください。やり方が分からない場合は、3-1 節を読み返してください。

```
void jButtonCopy_actionPerformed(ActionEvent e) {  
  
}
```

この { } 内に

上のテキストフィールド (jTextFieldFrom) に入力した文字列を下のテキストフィールド(jTextFieldTo)にコピーする

処理を記述すれば良いのです。前節で学習した通り、setText()メソッドを用いると、この処理はとりあえず次のような文になるはずです。

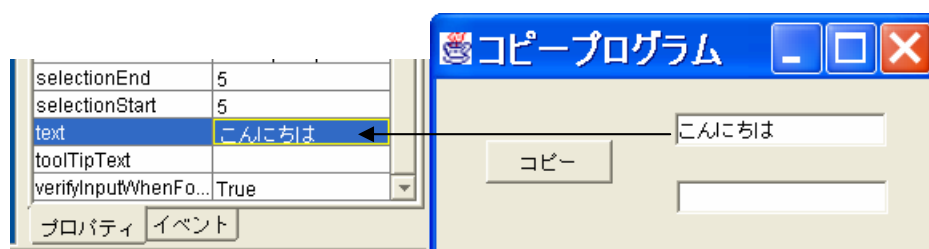
```
jTextFieldTo.setText( jTextFieldFromに入力された文字列 );
```

すると問題は、

「jTextFieldFromに入力された文字列」をどのように表すことができるか？

という事になります。

さて、「テキストフィールド」コンポーネントの「text」プロパティに入力した文字列はテキストフィールドに表示される、事はすでに学習しました。逆に、プログラム実行時に「テキストフィールド」コンポーネントに入力された文字は、下のように「text」プロパティに記憶（保存）されます。このように「text」プロパティは文字が保存される“場所”と考えることができます。



すると、テキストフィールドに入力された文字列とは、「text プロパティに入っている値」であることになります。その値を取り出す命令（メソッド）は **getText()** です。このメソッドを用いると

jTextField の text プロパティに入っている文字列 → **jTextField.getText()**

と表せます。

注意

- ① **getText()** の場合、() 内に何も記入しませんが、() そのものは必要です。これを省略するとエラーになるので注意して下さい。
- ② 前節で、文字列にするには " で囲むと説明しました。例えば「abc」を文字列にする場合には "abc" と表します。しかし、上の「jTextField.getText()」は、すでに これ自身が文字列 なので、" " で囲む必要はありません。もし、「jTextField.getText()」と表すと、" " で囲まれた部分が（Java 言語の命令ではなく単なる文字列と解釈され）「jTextField.getText()」という文字列としてそのまま表示されてしまいますので注意して下さい。

このことを用いて、次の空欄を埋めてコピープログラムを完成させてください。

```
void jButtonCopy_actionPerformed(ActionEvent e) {  
  
    jTextFieldTo.setText(  .getText());  
  
}
```

完成したら、プログラムを実行して動作を確認してください。

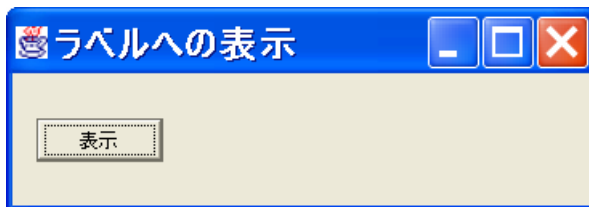
3-3 イベントとイベントハンドラの練習

【基礎課題 3-3-1】

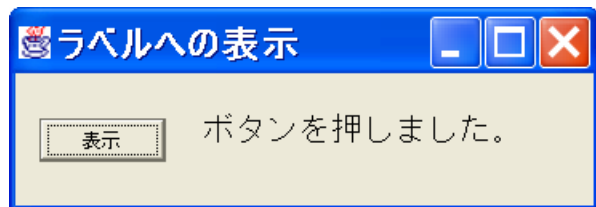
`setText()` メソッド（および `getText()` メソッド）は、`text` プロパティを持つコンポーネントについては全て定義されています。ですから、例えば、プログラム実行時にラベルコンポーネントに文字列を表示させる事もできます。

次のようなプログラムを作成して下さい。

プログラム起動時



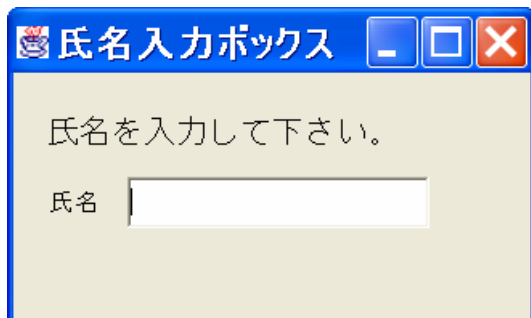
ボタンをクリックすると次のように表示される。



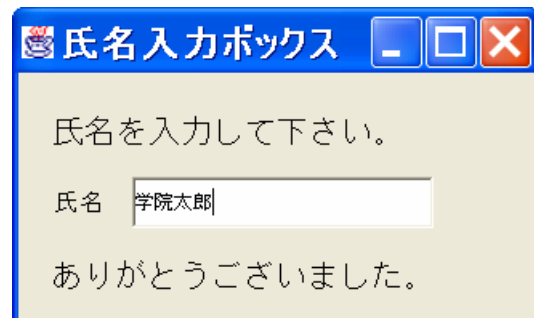
【基礎課題 3-3-2】

3-2 節で学んだ通り、ボタンコンポーネントについての `actionPerformed` とは、ボタンがクリックされた状態でした。アクションの種類は、コンポーネントによって異なります。例えばテキストフィールドコンポーネントの場合、文字列を入力して `[Enter]` キーを押した状態が `actionPerformed` の状態になります。このことを用いて、次のようなプログラムを作成してください。

プログラム起動時



氏名入力後、`[Enter]` キーを押すと、「ありがとうございました。」と表示される。



【基礎課題 3-3-3】

ここまでくれば勘のいい人は、

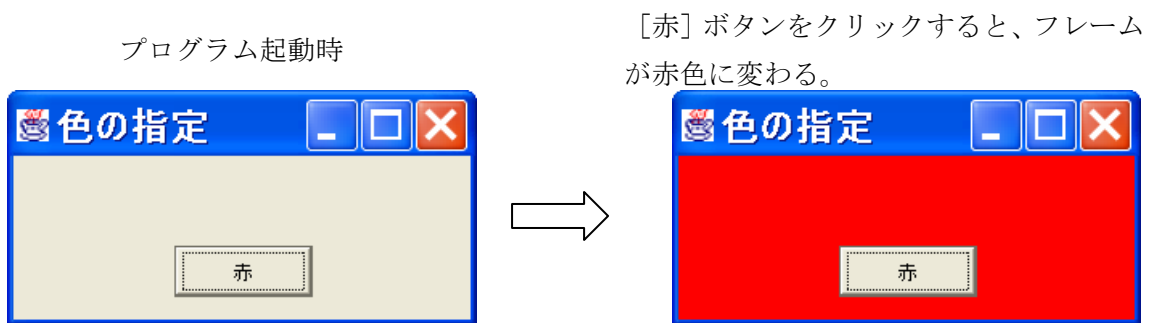
コンポーネントの各プロパティに対して「set...」や「get...」のメソッドがありそうだ。

という事に気づいたのではないのでしょうか？ そして

これらメソッドを用いれば、実行時にプロパティの値の取得や変更が可能だ。

というイメージを持つことができた人もいると思います。実際、Java 言語のコンポーネントはそうになっています。このように、いくつかの事例から一般的なルールを類推できた人は、プログラマとして大変有望です。

さて、本章の最後の課題として、フレームの色を実行時に変更してみましょう。まず、次のようなプログラムを作成しましょう。



プログラムの作成に当たってポイントとなるのは次の点です。

- ① 色を指定するのは、**background** プロパティでした。
- ② フレームの色を変える操作は、**contentPane** コンポーネントに対して行います。一般にフレームに対する操作は **contentPane** が引き受けています。
- ③ したがって、**contentPane** の **setBackground()** メソッドを用いれば、色を変更できそうです。

以上を念頭において、ボタンクリック時のイベントハンドラを作成すると次のようになります。なお、ボタンコンポーネントの **name** プロパティを「**jButtonRed**」としています。

```
void jButtonRed_actionPerformed(ActionEvent e) {  
    contentPane.setBackground(Color.red);  
}
```

赤色を指定していると思われる「**Color.red**」が新しく出てきた（見慣れない）部分で

すが、とりあえず、このプログラムを作成し、プログラムを実行させてみてください。
うまく行けば、ボタンクリック後、フレームの色が変わるはずです。

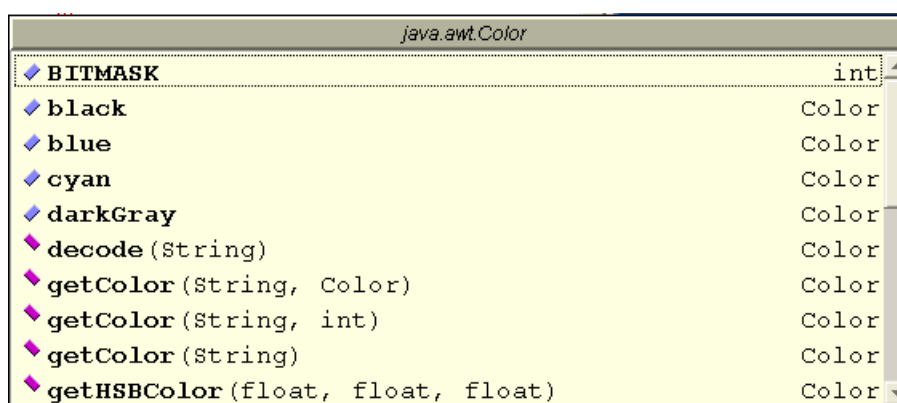
さて、実は **Color** は **Java** 言語が色に関する操作のために用意してくれている**クラス**です。
これは特別なクラスで、同じ名前で**オブジェクト**になっています。まずは、以下の2点をおさ
えておきましょう。

< Color オブジェクトについて >

- ① **Java** 言語では、**Color** というオブジェクトを用意しており、それには色の種類（プロパティ）や、色に対する操作（メソッド）が定義されています。
- ② 色の指定は、**Color.red** などのように指定します。どのような色を指定できるかは、プログラム作成時に、

Color.

と入力した時点でしばらく待つと、



というメニューが現れるので、そこから選択すれば良いのです。これは、**Color** オブジェクトに定義されているメソッドとプロパティ（メンバーと総称されます）のリストであり、**JBuilder** が用意している**コード補完機能**というものです。大変便利な機能なので、プログラム作成時には積極的に利用しましょう。

それでは、上で作成したプログラムを拡張して、今度は次のように「赤」、「青」、「黄」の3つのボタンを用意し、それぞれのボタンをクリックすると（フレームが）該当する色に変わるようにしてください。

