

第8章 CG 入門

【学習内容とねらい】

本章では、Java 言語を用いた CG（コンピュータグラフィックス）の描き方を学習します。Java 言語では、Graphics クラスに CG 作成に必要なメソッドが用意されており、それらを利用するだけで簡単に CG を作成することができます。その、“簡単に CG を描ける”という体験を（課題プログラムの作成を通じて）してもらい、ということが本章のねらいです。ですから、少し突っ込んだ CG の理論に関する内容には触れていません。それらについては、3年次に開講される、「CG プログラミング論」に譲ります（CG プログラミング論を履修する人は、この第8章の理解が必要になります）。

以上のような理由で、本章で扱う題材は基本的なものに限定しています。少し物足りないと思った人は、ぜひ、市販のテキスト等でより本格的な知識・技術を身につけてください。Java 言語を用いてかなり本格的な CG あるいは CG アニメーションを作成できる事に気づくはずですよ。少し努力すれば、皆をあっという間に驚かせる CG を作成することができますよ。

それでは、前置きはこの位にしてさっそく学習に取りかかりましょう。ともかく本章では、自分のプログラムで CG を描く楽しさを味わって下さい。

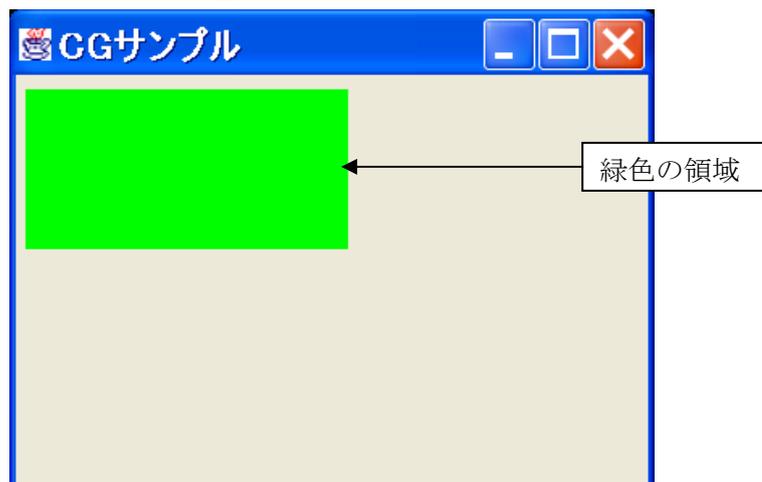
<本章の構成>

- | | |
|-----|------------------------------------|
| 8-1 | CG の描き方①—標準的な方法 (paint メソッドを使用) |
| 8-2 | CG の描き方②—getGraphics () メソッドを用いる方法 |
| 8-3 | 定型図形の描画 |
| 8-4 | 任意の多角形の描画 |
| 8-5 | 点画の描画 |
| 8-6 | 画像ファイルの読み込みと表示 |
| 8-7 | 描画処理の改良—再描画処理 |

8-1 CGの描き方①ー標準的な方法 (paintメソッドを使用)

【応用課題 8-1-A】

まず手始めに、次のような CG を描いてみましょう。それは、実行すると、次のようにフレームの左上の矩形領域が緑色に塗られるというプログラムです。



市販のテキスト等で説明されている最も標準的なCG作成方法は、コンポーネントに備わっている**paint**メソッド (内) に、CG作成処理を記述することです。paintメソッドは主立ったコンポーネントに用意されており、そのコンポーネントが生成された瞬間に実行されます。ですから、paintメソッドを用いると、プログラム実行と同時に (そこに記述された処理が) 実行されるようになっています。また、別のWindowsが重なり、一部分 (あるいは全部) が隠れてしまった後に、再描画する際にも自動的に呼び出されます。

では、アプリケーションを新規作成し、ソースビューを開いて下さい。そして、次ページのように、プログラム上方の「フレームのビルド」とコメントがある部分 (Frame1 クラスのコンストラクタ) を見つけ、その下に、同じく次ページのように paint メソッドを書き加えて下さい。枠内が新たに記述した部分です。

さて、プログラムの解説に進む前に、少しpaintメソッドについて補足しておきましょう。今定義しているFrame1 クラスはJFrameクラスを継承して作られたものです。そしてJFrameクラスにはpaintメソッドが定義されており、その処理内容は、フレーム全体をbackgroundカラーで塗りつぶすというものです。そこで、それ以外の描画処理が必要になったときには、次ページのように、paintメソッドを書き換えるのです。すると、今度は書き直されたpaintメソッドが有効になります。7-4 節で説明したように、継承したサブクラスで (スーパークラスの) メソッドの内容を書き換えることをメソッドの**オーバーライド** (書き換え) と言います。以上を念頭に置いて、プログラムの作成に進んで下さい。

```
//フレームのビルド
```

```
public Frame1() {  
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);  
    ...  
}
```

新たに挿入した部分

```
public void paint(Graphics g) {  
    g.setColor(Color.green);  
    g.fillRect(10, 50, 200, 100);  
}
```

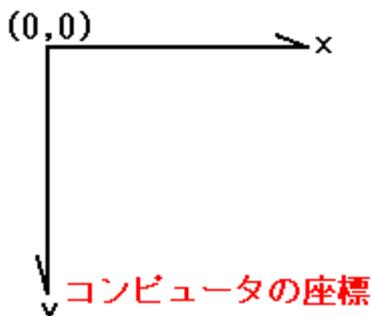
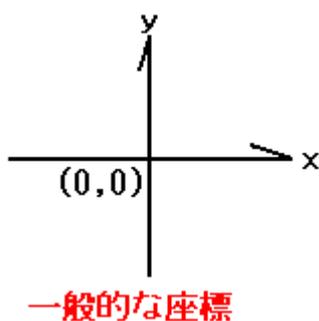
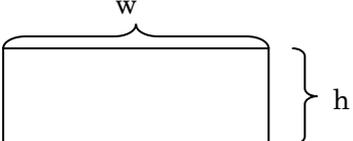
```
//コンポーネントの初期化
```

```
private void jbInit() throws Exception {  
    ...  
}
```

<解説>

- ① paint メソッドの引数は Graphics クラスの変数 (オブジェクト) です。上の例では g という名前にしています。これで、対象とするコンポーネント (今の場合フレーム) の Graphics オブジェクトを取得できます。
- ② setColor () メソッドは、Graphics クラスに用意されているメソッドで、CG を作成する際の色を指定します。今の場合、緑色です。Color オブジェクトについては、第3章 (p.60) でも説明しました。
- ③ fillRect (x, y, w, h) メソッドは、点 (x, y) を左上頂点とし、横幅 w、縦の高さ h の四角形を描き、その内部を指定色で塗りつぶす処理を行います。

座標の取り方については、下を参照して下さい。 (x, y)



CGの世界では、左上を原点 (0, 0,)とし、y座標が下向きに伸びていることに注意して下さい。

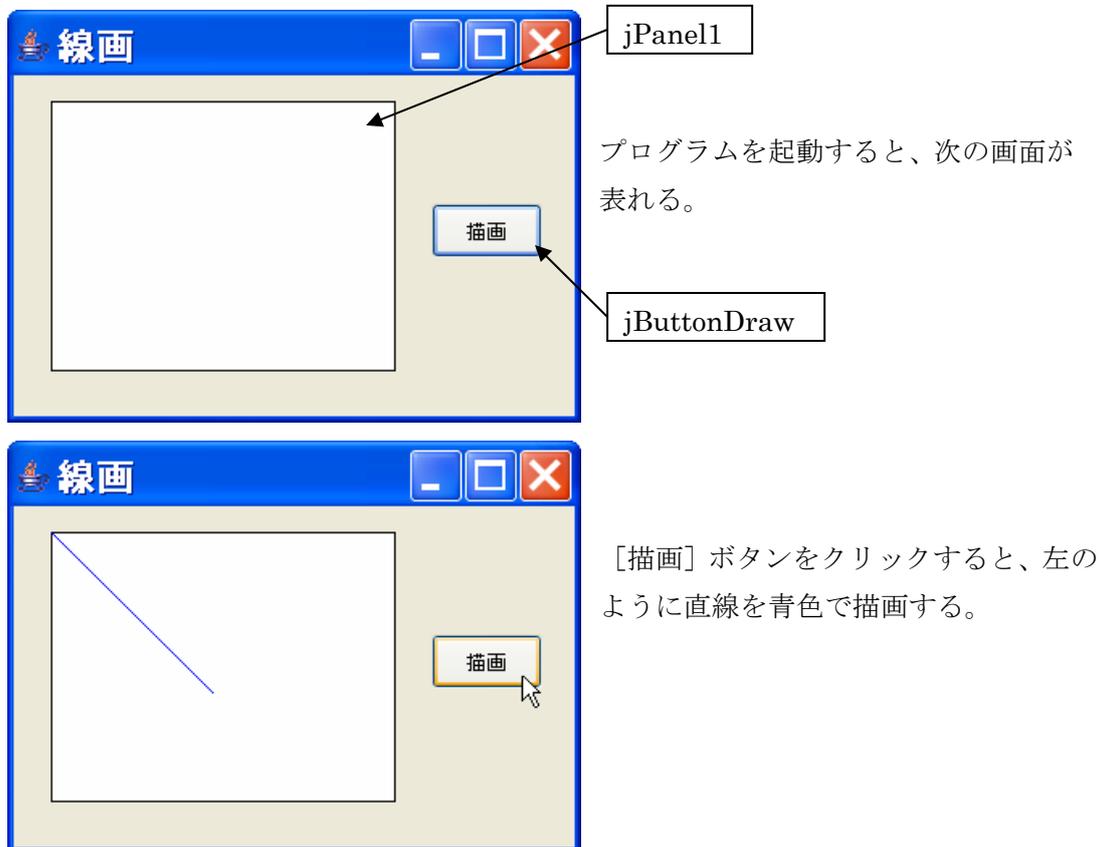
作成したら、プログラムを実行し動作を確認して下さい。

8-2 CGの描き方②ーgetGraphics()メソッドを用いる方法

前節の `paint` メソッドを用いる方法は簡単なのですが、プログラムの実行と同時に描画されてしまいます。そこで、例えばボタンを押したときに CG 描画を開始させたい場合など、CG 作成の動作制御を行う際には少し不便です。そこで、以下では、対象とするコンポーネントの `Graphics` オブジェクトを直接取得する `getGraphics()` メソッドを用いる方法を用いることにします。

【応用課題 8-2-A】

次のようなプログラムを作りましょう。



なお、パネルコンポーネントの `background` プロパティは白色にしてください。

この [描画] ボタンのイベントハンドラは次のようになります。

```

void jButtonDraw_actionPerformed(ActionEvent e) {
    Graphics g=jPanell1.getGraphics(); //Graphics オブジェクトの取得 ①
    g.setColor(Color.blue);          ②
    g.drawLine(0,0,100,100);         ③
    g.dispose(); //Graphics オブジェクトの解放 ④
}

```

<解説>

- ① コンポーネントの **Graphics** オブジェクトを取得するには、`getGraphics()` メソッドを用います。1行目では、パネルコンポーネントの **Graphics** オブジェクトを、**Graphics** 型変数（オブジェクト）`g` に初期値として与えています。
- ② 2行目で使用する色を青色にしています。
- ③ 3行目の `drawLine(x1,x2,y1,y2)` は、2点 $(x1,y1)-(x2,y2)$ を結ぶ直線を描くメソッドです。
- ④ **dispose** とは「処分する」という意味ですが、今の場合 **Graphics** オブジェクト `g` をメモリから解放（消去）することを意味します。①の様に `getGraphics()` メソッドを用いて独自に **Graphics** オブジェクトを取得した場合、使用后（描画後）その **Graphics** オブジェクトを廃棄（メモリから解放）することが推奨されています。

補足 小さなプログラムの場合、④を行わなくても実際上の支障はありませんが、コンピュータのメモリ容量は限られているので④を記述した方が無難です。なお、8-1節で説明した `paint` メソッドの引数 `g` については、**Java** 実行環境が管理しているので、`dispose` メソッドで解放してはいけません。

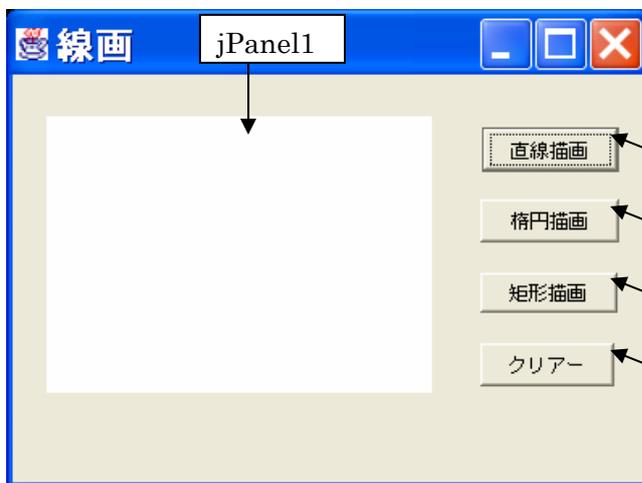
- ※ 今の場合、**Graphics** オブジェクト `g` が（フレームではなく）パネルコンポーネントのオブジェクトなので、CG 作成をパネル上で行っていることに注意して下さい。このように `getGraphics()` メソッドを用いれば、ボタンやラベルの上など、様々なコンポーネントの **Graphics** オブジェクトを取得でき、したがって当該コンポーネント上での **CG** 作成を容易に行えます。

8-3 定型図形の描画

矩形（長方形や正方形）や円（楕円）については、それらを描画するメソッドが Java 言語には備わっています。その使い方を本節で練習しましょう。

【練習課題】

次のようなプログラムを作成しましょう。



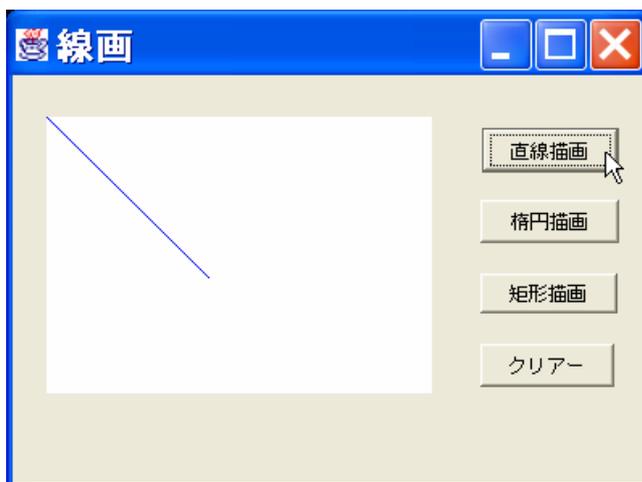
起動すると次のような画面が現れる。

jButtonLine

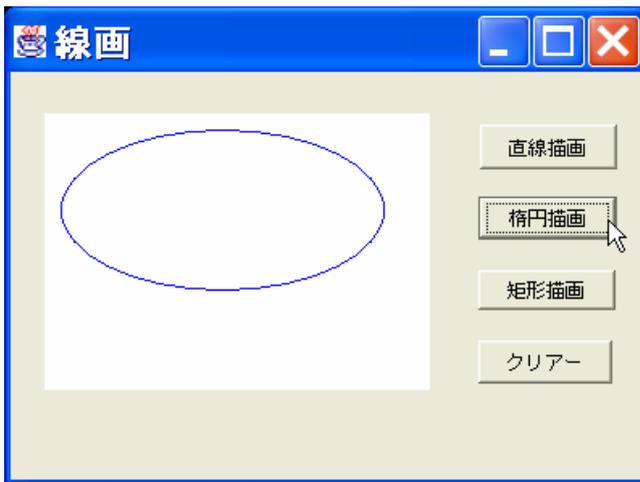
jButtonOval

jButtonRect

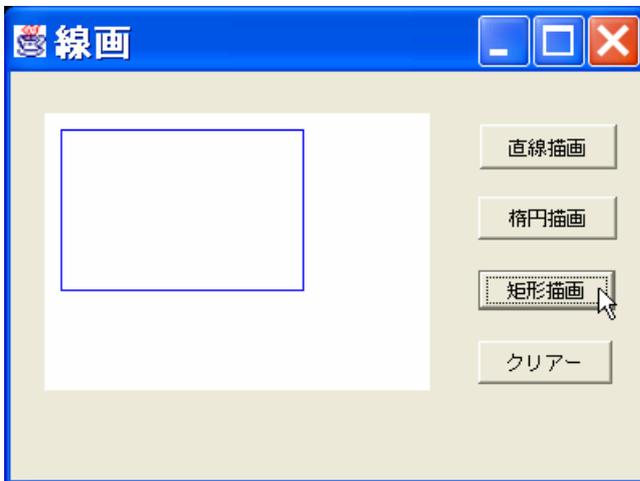
jButtonClear



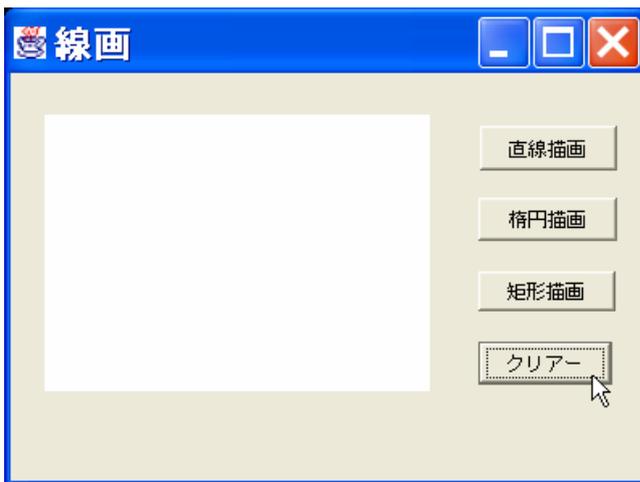
[直線描画] ボタンをクリックすると直線を描く。



[楕円描画] ボタンをクリックすると楕円を描く。



[矩形描画] ボタンをクリックすると、四角形を描く。



[クリアー] ボタンをクリックすると、全ての画像を消去する。

各ボタンのイベントハンドラは次の通りです。これを眺めれば意味は大体分かります。正確な意味はプログラムの後にある<解説>で確認して下さい。

<[直線描画]ボタン>

```
void jButtonLine_actionPerformed(ActionEvent e) {
    Graphics g=jPanell1.getGraphics();
    g.setColor(Color.blue);
    g.drawLine(0,0,100,100);
    g.dispose();
}
```

<[楕円描画]ボタン>

```
void jButtonOval_actionPerformed(ActionEvent e) {
    Graphics g=jPanell1.getGraphics();
    g.setColor(Color.blue);
    g.drawOval(10,10,200,100);    ①
    g.dispose();
}
```

<[矩形描画]ボタン>

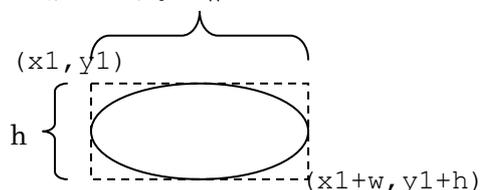
```
void jButtonRect_actionPerformed(ActionEvent e) {
    Graphics g=jPanell1.getGraphics();
    g.setColor(Color.blue);
    g.drawRect(10,10,150,100);    ②
    g.dispose();
}
```

<[クリア]ボタン> ⑤

```
void jButtonClear_actionPerformed(ActionEvent e) {
    int XMax=jPanell1.getWidth(); //パネルの幅の取得    ③
    int YMax=jPanell1.getHeight(); //パネルの高さの取得    ④
    Graphics g=jPanell1.getGraphics();
    g.setColor(Color.white);
    g.fillRect(0,0,XMax,YMax); //領域内を指定色で塗りつぶす
    g.dispose();
}
```

<解説>

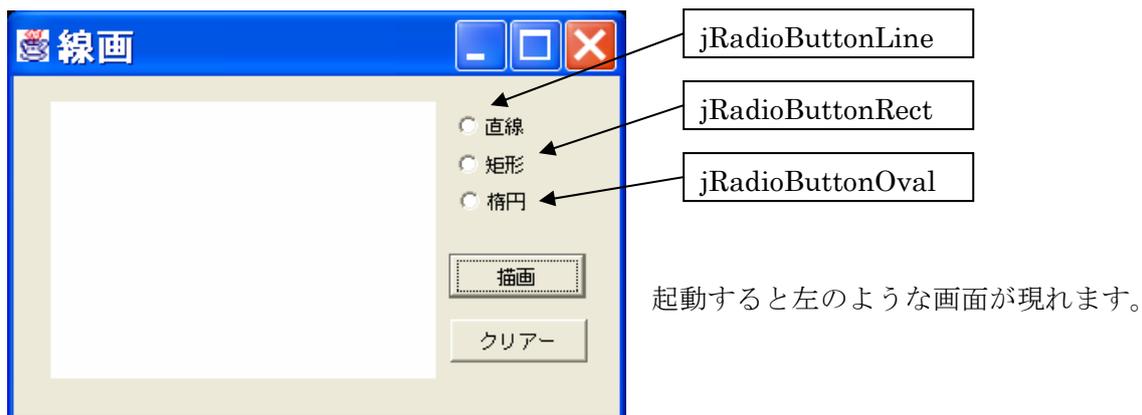
- ① `drawOval(x1, y1, w, h)` は下のように、 $(x1, y1)$ を左上隅として、幅 w 、高さ h の四角形に内接する楕円を描きます。

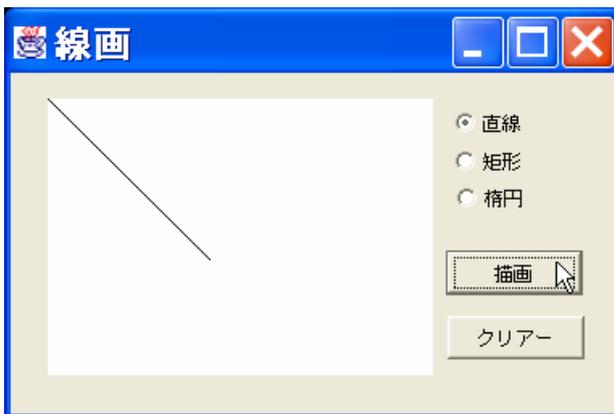


- ② `drawRect(x1, y1, w, h)` メソッドは、左上の頂点を $(x1, y1)$ とし、幅 w 、高さ h の四角形を描きます。
- ③ `getWidth()` メソッドは、対象とするオブジェクト（今の場合パネルコンポーネント）の横幅の値を与えます。
- ④ `getHeight()` メソッドは、対象とするオブジェクト（今の場合パネルコンポーネント）の（縦方向の）高さを与えます。
- ⑤ [クリア] ボタンのイベントハンドラの処理内容は、パネル領域を白色で塗りつぶすということです。上の例から分かる通り、「draw...」メソッドを「fill...」に変えると図形内の色を塗りつぶす処理を行います。

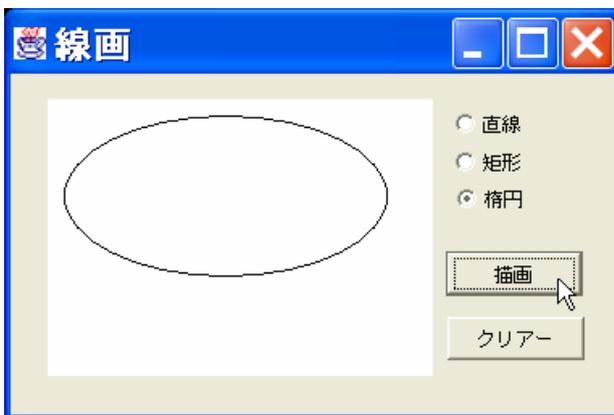
【応用課題 8-3-A】

上の【練習課題】のプログラムを改良して、次のようなプログラムを作成しましょう。作成に当たっては次ページの指示に従って下さい。





「直線」欄をクリックして [描画] ボタンをクリックすると、直線を描きます。



「楕円」欄をチェックして [描画] ボタンをクリックすると、楕円を描きます。矩形についても同様です。また、[クリアー] ボタンをクリックすると図形を消去します。

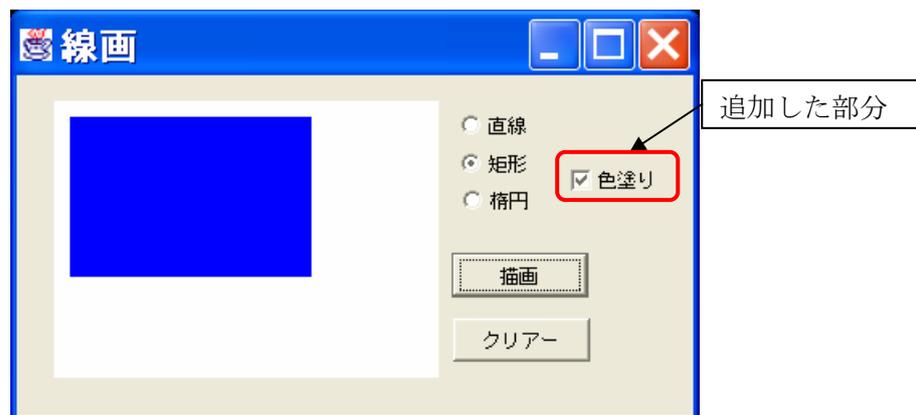
[描画] ボタンのイベントハンドラを次のように記述しました。

```
void jButtonDraw_actionPerformed(ActionEvent e) {
    Graphics g=jPanell.getGraphics();
    if(jRadioButtonLine.isSelected()) {
        LineDraw(g); //線を描くメソッドを呼び出す
    }
    else if(jRadioButtonRect.isSelected()) {
        RectDraw(g); //四角形を描くメソッドを呼び出す
    }
    else {
        OvalDraw(g); //楕円を描くメソッドを呼び出す
    }
    g.dispose();
}
```

上で示した処理を実現するよう、メソッド LineDraw(g)、RectDraw(g)、OvalDraw(g) を定義して下さい。矩形や楕円の大きさは、【練習課題】の通りで結構です。

【応用課題 8-3-B】

【応用課題 8-3-A】のプログラムに、次のように（図形内の）色を塗りつぶす選択欄を付けて加えて下さい。例えば、「矩形」および「色塗り」欄をチェックして [描画] ボタンをクリックすると、色が塗りつぶされた四角形が描画されます。



※ 直線に関しては、色塗り欄は無効なので（チェックの有無は関係ないので）、以前のプログラムから変更はありません。

【応用課題 8-3-C】

次のような、市松模様（隣り合う四角形領域の色が異なる模様）を描くプログラムを作成して下さい。下の例では、白色と赤色で塗り分けています。

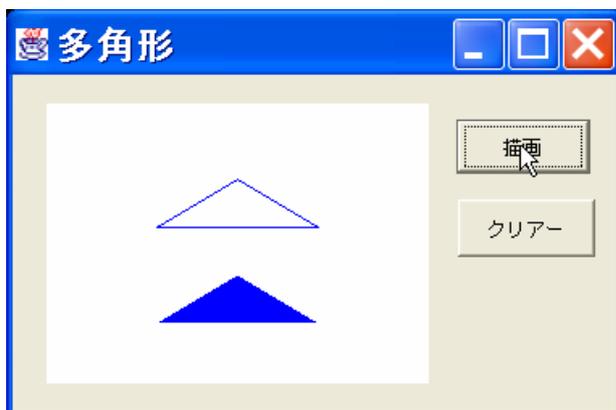


8-4 任意の多角形の描画

本節では、任意の多角形を描く `drawPolygon` メソッドの使い方を学習しましょう。

【練習課題】

次のように、三角形を描くプログラムを考えましょう。



プログラムを起動し、[描画] ボタンをクリックすると、二つの三角形（下方は内部の色を塗りつぶしている）を描く。

このときの [描画] ボタンクリック時のイベントハンドラは、次のようになります。

```
void jButtonDraw_actionPerformed(ActionEvent e) {
    Graphics g=jPanell1.getGraphics();
    int xc=jPanell1.getWidth()/2; //パネルの x 座標の中心を求める
    int yc=jPanell1.getHeight()/2; //パネルの y 座標の中心を求める
    int x[]=new int[3]; //大きさ3の配列 x の宣言
    int y[]=new int[3]; //大きさ3の配列 y の宣言
    x[0]=xc-50; x[1]=xc; x[2]=xc+50; //三角形の3点の x 座標を配列に入れる
    y[0]=yc-10; y[1]=yc-40; y[2]=yc-10; //同じく3点の y 座標を配列に入れる
    g.setColor(Color.blue); //描画色を青色に指定
    g.drawPolygon(x,y,3); //多角形（今の場合三角形）の描画
    y[0]=yc+50; y[1]=yc+20; y[2]=yc+50; //y 座標の更新（下に60だけずらす）
    g.fillPolygon(x,y,3); //多角形（今の場合三角形）の描画および塗りつぶし
    g.dispose();
}
```

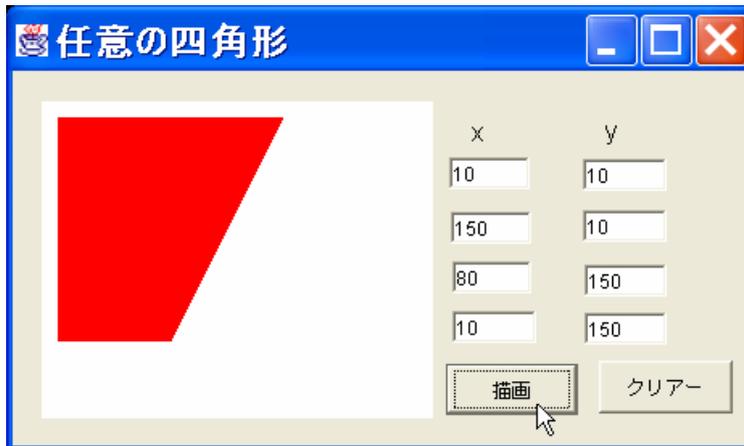
<解説>

- プログラムの大まかな意味は、コメントから分かると思います。
- `drawPolygon(x,y,n)` メソッドは、`x` 座標配列、`y` 座標配列、そして点の個数（配列の大きさ）を引数として指定します。配列については、4-12 節を参照して下さい。
- そして、このメソッドが呼び出されると、点 $(x[0],y[0])$ から点 $(x[1],y[1])$ 、・・・点 $(x[n-1],y[n-1])$ と順に直線で結び、最後に再び点 $(x[0],y[0])$ に戻ってきます。これにより、`n` 点を結ぶ多角形が描画されます。
- `fillPolygon(x,y,n)` メソッドの場合は、線で結んだ多角形の内部を指定した色で塗りつぶします。

作成したら実行し、動作を確認して下さい。

【応用課題 8-4-A】

上の Polygon メソッドを用いて、次のように、任意の四角形を描いてその内部を赤色で塗りつぶすプログラムを作成して下さい。



4 点の `x`、`y` 座標を入力して [描画] ボタンをクリックすると、4 点を結び、その内部を赤色で塗りつぶした四角形を描く。

8-5 点画の描画

前節までの学習で、任意の多角形を描画することができるようになりました。しかし、多角形では表現しにくい、より一般的な図形はどのように描画すればよいのでしょうか？それは、画面の最小単位の点（ピクセルと言います）毎に指定色で色づけすればよいのです。と言うのは、どんなに複雑な図形でも点の集まりで表現できるからです。そこで、本節では、そのような点画を描く練習をしてみましょう。と言っても新しいメソッドを用いるわけではありません。実は、すでに学習した `drawLine(x1,y1,x2,y2)` メソッドを用いて、2点の座標を同じにすると、（当然ですが）その点に指定色で点を打つ命令になるのです。次の課題で、`drawLine` メソッドを用いて、少し面白い点画を描いてみましょう。

【応用課題 8-5-A】 (chaos)

今、ある点列 $(X_1, Y_1), (X_2, Y_2), (X_3, Y_3), \dots, (X_i, Y_i), \dots, (X_n, Y_n)$ を考えます。ここに、 (X_i, Y_i) が決まったときにその次の (X_{i+1}, Y_{i+1}) は次の式で決まるものとします。

$$X_{i+1} = Y_i - 0.97 \times X_i + 5 / (1 + X_i^2) - 5, \quad Y_{i+1} = -0.995 \times X_i$$

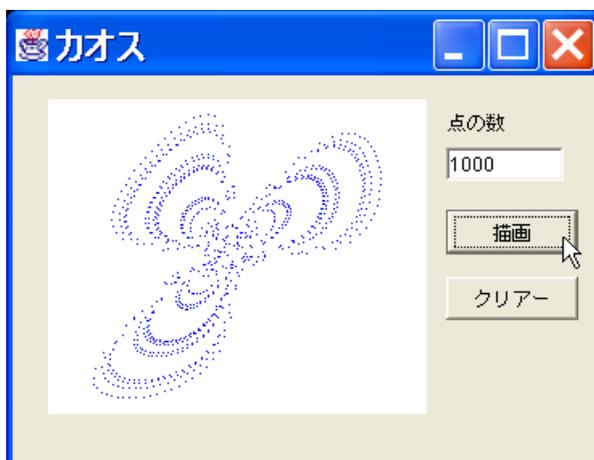
例えば、 $(X_1, Y_1) = (1, 0)$ とすると、

$$X_2 = Y_1 - 0.97 \times X_1 + 5 / (1 + X_1^2) - 5 = -0.97 + 5/2 - 5 = -3.47$$

$$Y_2 = -0.995 \times X_1 = -0.995$$

と (X_2, Y_2) が求まり、同様に、 $(X_3, Y_3), \dots, (X_n, Y_n)$ が次々と求まって行きます。

さて、最初の点を $(X_1, Y_1) = (1, 0)$ として上の要領で点列を求め、順に表示させて行くと次のように鳥が羽を広げたような不思議な図形が現れます。このプログラムを作成してください。



点の数を 1000 にして描画したところ
(点の色は青にしています。)



点の数を 30000 にして描画したところ

ヒント

- ① 点の数を N とすると、上の処理は、点 (X_i, Y_i) を表示させるという処理を、 $i=1 \sim N$ について N 回繰り返すことで実現できます。
- ② 上の (X_i, Y_i) の値は小さな数になるので見えにくくなります。そこで実際には何倍かに拡大する必要があります。上の例では $(6 * X_i, 6 * Y_i)$ と 6 倍に拡大 して表示しています。
- ③ 座標の原点はパネルコンポーネントの中心となるようにしています。つまり、原点の位置をずらしています（座標の原点をパネルの左上隅のままにしておくと、マイナスの値を表示することができないので注意して下さい）。
- ④ パネルコンポーネントの幅と高さは 200 程度にしています。
- ⑤ (X_i, Y_i) が求まるたびにその点（実際には 6 倍に拡大した座標）に点を打てば良いのです。その際、 (X_i, Y_i) は実数なので、drawLine メソッドの引数にするためには、整数型に型キャストする必要があります。

上の課題プログラムについては、プログラミングの HP

<http://ext-web.edu.sgu.ac.jp/HIKO/Prog>

の該当部分に、クラスファイルが入ったフォルダを `chaos.exe` という名前の自己解凍形式フォルダとして掲載しています。これをダウンロードしてダブルクリックして下さい。すると、自動的に解凍されフォルダ `chaos` が現れます。このフォルダ内のクラスファイルを用いれば、プログラムを実行し動作結果を確認する事ができます。各自、学習の際の参考にして下さい。クラスファイルからの実行の仕方は、巻末の「付録A クラスファイルからの実行の仕方」を参照して下さい。

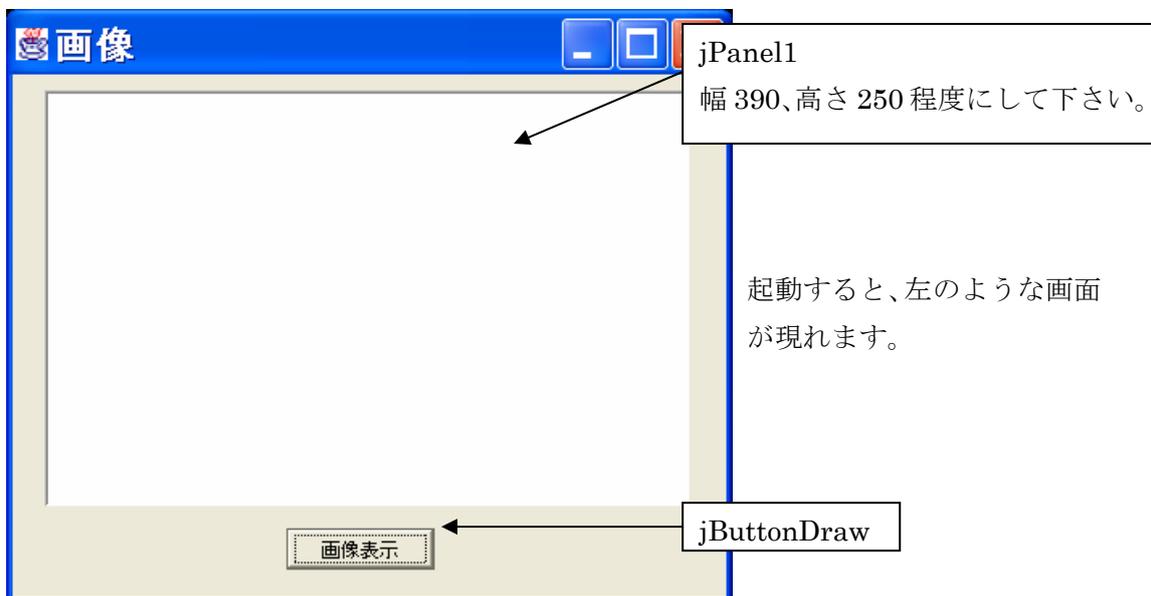
8-6 画像ファイルの読み込みと表示

前節までは、プログラムで描画を行う方法を学習しましたが、デジタルカメラの写真などファイルに保管されているデジタル画像については、画像ファイルから読み込んでそれを適当なコンポーネント（例えばパネルコンポーネント等）に表示させることができます。本章の最後に、その方法を学習しましょう。

Java言語では、{gif,jpg,png} 形式の画像を読み込む機能があります。「プログラミング」のHPの該当部分に掲載している自己解凍形式のフォルダFig.exeをダウンロードし、それを解凍して下さい。フォルダFigの中には {Falcon.jpg, Grappa.jpg, Desk.jpg, Concert.jpg, Milano.jpg} の5つのサンプル画像を納めています。以下では、これを入力画像ファイルとして用います。以下の課題では、作成するプロジェクトのフォルダ内にフォルダFigを必ずコピーしておいて下さい。

【練習課題】

次のようなプログラムを作ってみましょう。





画像表示ボタンをクリックすると、左のような画像が現れます。

この「画像表示」ボタンのイベントハンドラは次のようになります。

```
void jButtonDraw_actionPerformed(ActionEvent e) {
    Graphics g=jPanell1.getGraphics();
    ImageIcon Icon=new ImageIcon("Fig¥¥Falcon.jpg"); //指定したファイル
    //内の画像を ImageIcon オブジェクト Icon に読み込む
    Image Fig1=Icon.getImage(); //Image (画像) オブジェクトの取得 ②
    g.drawImage(Fig1,0,0,this); //Image オブジェクトの表示 ③
    g.dispose(); //Graphics オブジェクトの解放
}
```

<解説>

- ① まず、画像ファイル「Falcon.jpg」に保管された画像データを ImageIcon (イメージアイコン) クラスのオブジェクト Icon に読み込みます。上のように、ImageIcon クラスのコンストラクタは画像ファイル名を引数としてオブジェクトとして生成することで、画像データ ({gif,jpg,png} 形式のもの) を保管できます。

※ 上の例では、ファイル名として"Fig¥¥Falcon.jpg"と指定していますが、これは、Fig フォルダ内にある Falcon.jpg ファイルという意味です。本来は¥記号は一つでフォルダの区切りを意味するのですが、これは特別な意味を持つ制御記号でもあるので、文字列定数の中で¥を表したい場合は¥¥の様に二つ続けて記述しなければなりません。1文字では、コマンドだと解釈されてしまうので注意して下さい。なお、JBuilder が通常使用しているフォントでは、「¥」が「\」と表示されます。ですから「¥」とキーインしたのに「\」が表示されても、気にせず作業を続けて結構です。

- ② Java で画像データを表示させるためには、Image クラスのオブジェクトに変換しなければなりません。そのために、ImageIcon オブジェクト Icon から getImage () メソッドによって、Image オブジェクト Fig1 に変換させます。

※ このように Java 言語では、

画像データ→ImageIcon オブジェクトへ読み込み→Image オブジェクトへ変換
という形で、画像データを Image オブジェクトに変換します。

- ③ Image オブジェクトを表示するには **drawImage ()** メソッドを用います。例えば"A"という名前の Image オブジェクトを表示させるためには、次のように記述します。

```
drawImage (A, x, y, this)
```

ここに、(x, y) は表示させる画像オブジェクト (の左上隅) の表示位置です。第 4 引数については、通常は this と指定しておいて下さい。

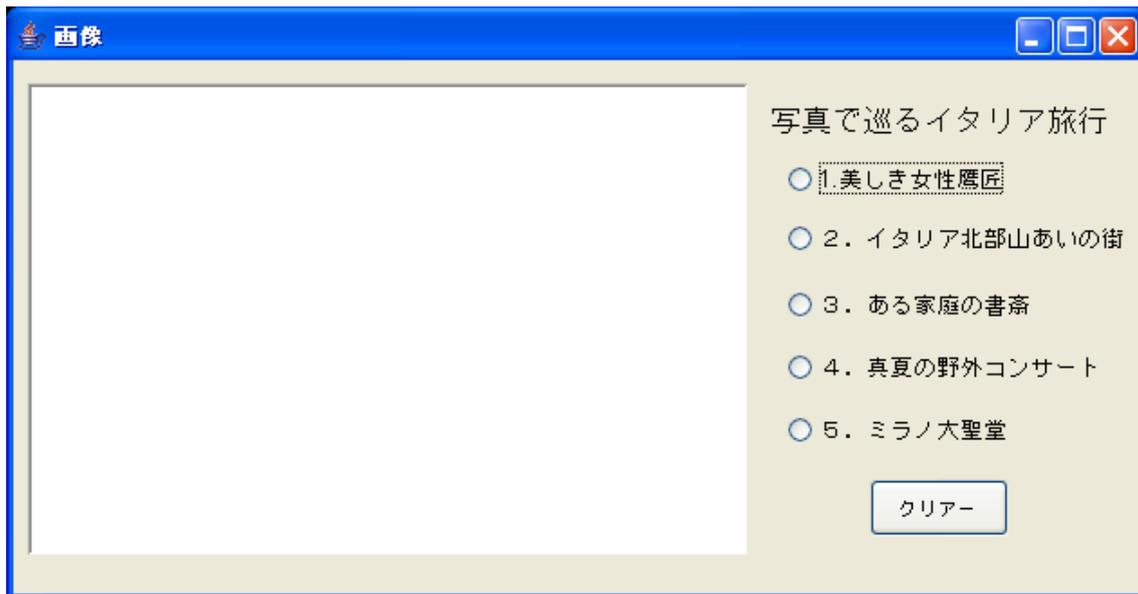
- ※ なお、用意した画像の大きさは、幅 390、高さ 250 程度なので、パネルの大きさもその程度の大きさにして下さい。パネルのサイズをこれより大きくすると、余白部分ができてしまいます。

プログラムを作成したら動作を確かめて下さい。

【応用課題 8-6-A】 写真で巡るイタリア旅行

画像表示の技術を使って、次のように、指定した画像を表示させるプログラムを作成して下さい（作成するプロジェクトのフォルダ内に、フォルダ Fig をコピーすることを忘れないで下さい）。

まず、プログラムを起動すると下の画面が現れます。

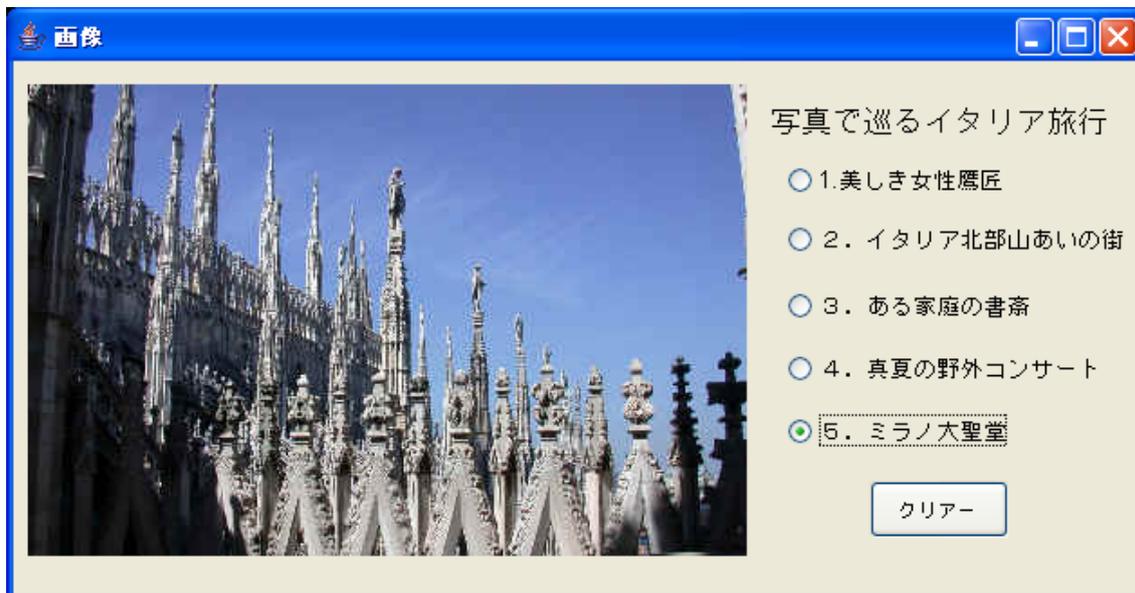


ここで、画像リストから、適当なラジオボタンをクリックすると、該当する画像が表示されます。

下は、1 を選択したところ。



下は、5を選択したところ。



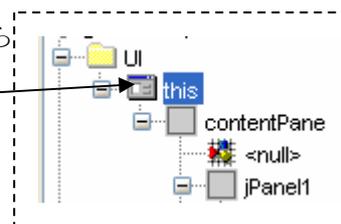
このプログラムでは、5つの画像を5つの Image オブジェクトに格納するようにします。そこで、下のように、Image オブジェクト (の配列) Fig[] を インスタンス変数として宣言して下さい。点線枠内が入力部分です。

```
public class Frame1 extends JFrame {  
    JPanel contentPane;  
    JPanel jPanel1 = new JPanel();  
    . . .  
    Image[] Fig=new Image[5]; //Image オブジェクトを配列型で宣言  
    //フレームのビルド
```

さて、このプログラムは プログラム開始時に (自動的に) 5つの画像が Image オブジェクトとして格納されるようにする必要があります。それにはどのようなイベントを用いれば良いか分かるでしょうか・・・? 幾つかの方法が考えられますが、ここではフレームが画面に表示 (生成) された瞬間に生ずる「componentShown」というイベントを利用しましょう。次の手順に従って作成してください。

<Image オブジェクト設定手順>

- ① 上のようなフレームを設計した後で、構造ペインから this (フレームのこと) を選択します。



- ② 次に、設計画面の右端にあるインスペクタの「イベント」欄から、componentShown 欄を選択（ダブルクリック）します。

ancestorMoved	
ancestorResized	
caretPositionCha...	
componentAdded	
componentHidden	
componentMoved	
componentRemo...	
componentResiz...	
componentShown	this_componentShc
focusGained	
focusLost	

- ③ 現れたソースエディタに次のように記述します（点線枠線分が記述した部分）。記述内容は【練習課題】(p.207)と同様なので理解できるはずです。これで5つの画像データを5つの要素を持つ Image オブジェクトの配列 Fig に格納する事ができました。

```
void this_componentShown(ComponentEvent e) {
    ImageIcon[] Icon= new ImageIcon[5]; //ImageIcon オブジェクトを配列
                                     として宣言
    //ImageIcon オブジェクトへの画像データの読み込み
    Icon[0]= new ImageIcon("Fig¥¥Falcon.jpg");
    Icon[1]= new ImageIcon("Fig¥¥Grappa.jpg");
    Icon[2]= new ImageIcon("Fig¥¥Desk.jpg");
    Icon[3]= new ImageIcon("Fig¥¥Concert.jpg");
    Icon[4]= new ImageIcon("Fig¥¥Milano.jpg");
    //ImageIcon オブジェクト → Image オブジェクトへの変換
    for (int i=0;i<=4;i++) {
        Fig[i]=Icon[i].getImage();
    }
}
```

続いてラジオボタンをクリックした時に該当する画像を（パネルに）表示するようにするには、ラジオボタンクリック時のイベントハンドラを次のように記述します。

<1番上のラジオボタンクリック時の処理>

```
void jRadioButton1_actionPerformed(ActionEvent e) {
    Graphics g=jPanell1.getGraphics(); //（パネルの）Graphics オブジェクト
                                     の取得
    int w=jPanell1.getWidth(); //パネルの幅の取得
    int h=jPanell1.getHeight(); //パネルの高さの取得
    g.drawImage(Fig[0],0,0,w,h,this); //最初（0番目）の画像の表示
    g.dispose();
}
```

以下、残りの4つのラジオボタンクリック時の処理についても同様に記述できるはずです。なお、上のプログラムにおいて、drawImage メソッドの引数が先程と変わっている事に気づいたと思います。実は、drawImage メソッドは

```
drawImage(Imageオブジェクト、x,y,w,h,this)
```

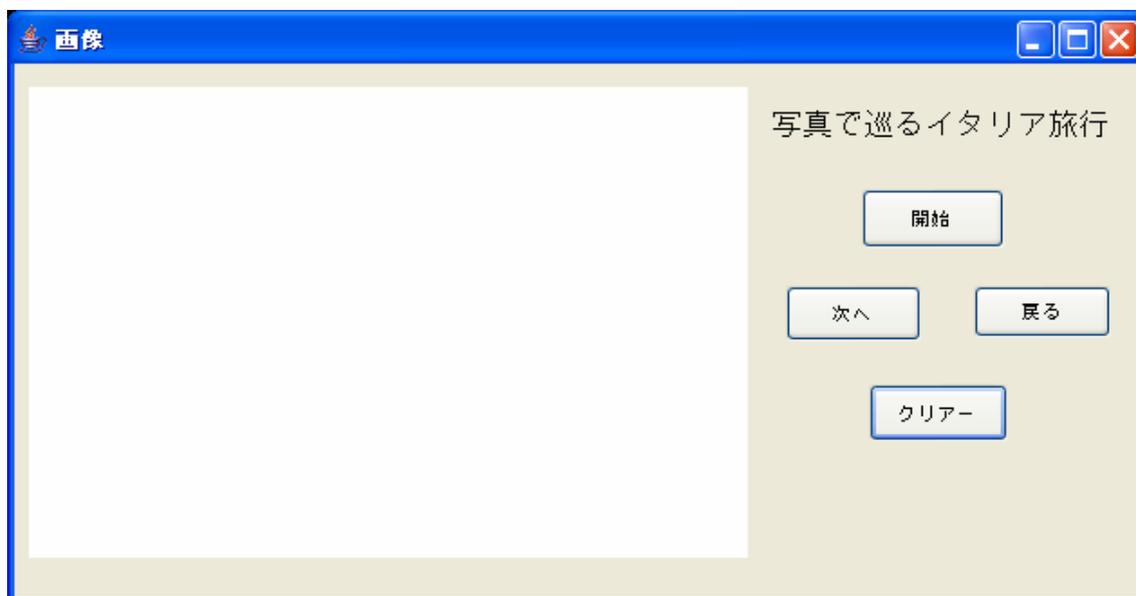
という形式で、指定した画像オブジェクトを、幅 w、高さ h の領域に収まるように（縮尺して）表示させる、という事もできます。これは大変便利なのですが、本来の画像の縦横サイズ比と指定領域のそれが大きくずれてしまうと、画像がひずんでしまうので注意が必要です。

それでは、[クリアー] ボタンの処理も含めてプログラムを完成させ、動作を確認して下さい。なお、このプログラムを実行するには、プロジェクトのフォルダ内にフォルダ Fig をコピーしておくことを忘れないように注意して下さい。

【応用課題 8-6-B】 スライドショー

【応用課題 8-6-A】 を改良して、次のように5枚の画像を（順番に）次々と表示するプログラムを作成して下さい。

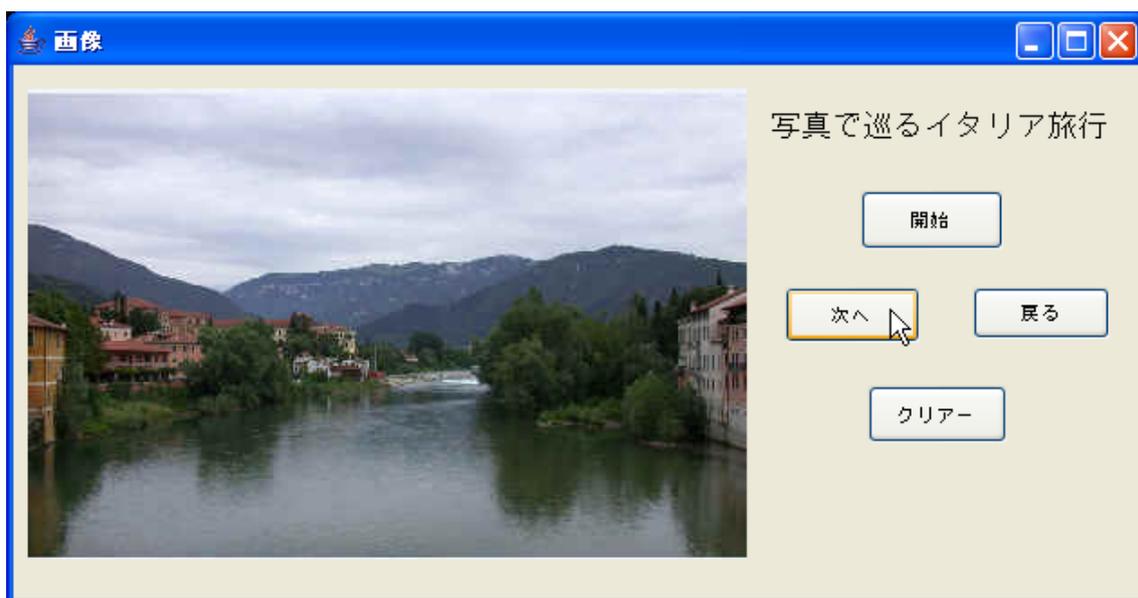
プログラムを起動すると、下のような画面が現れます。



ここで、まず [開始] ボタンをクリックすると、最初の画像（【応用課題 8-6-A】 の1の画像）が表示されます。

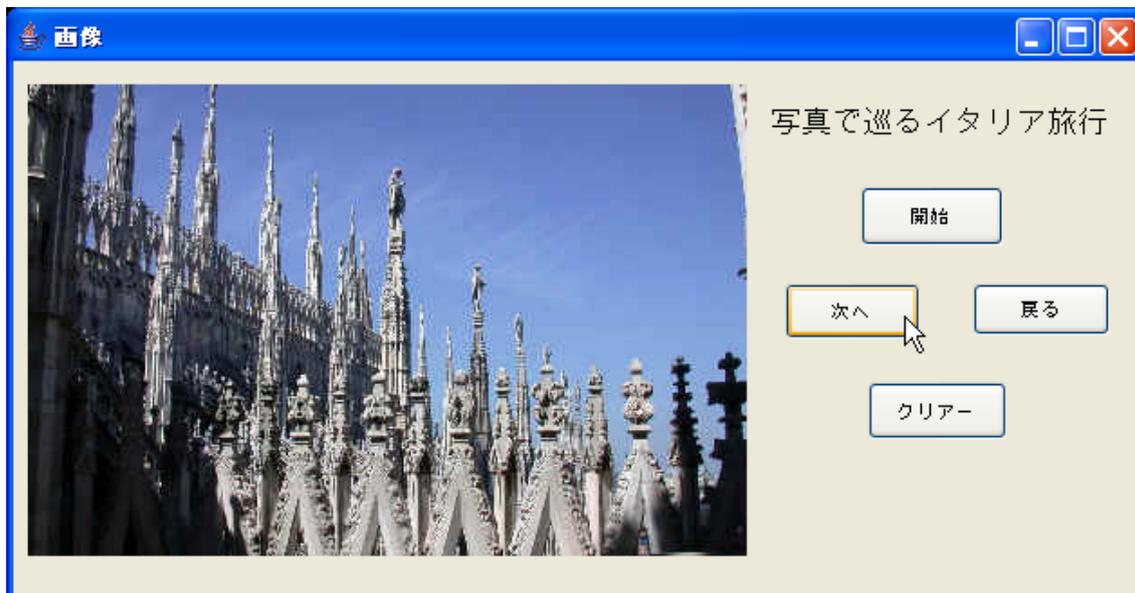


[次へ] ボタンをクリックすると、2番目の画像が表示されます。

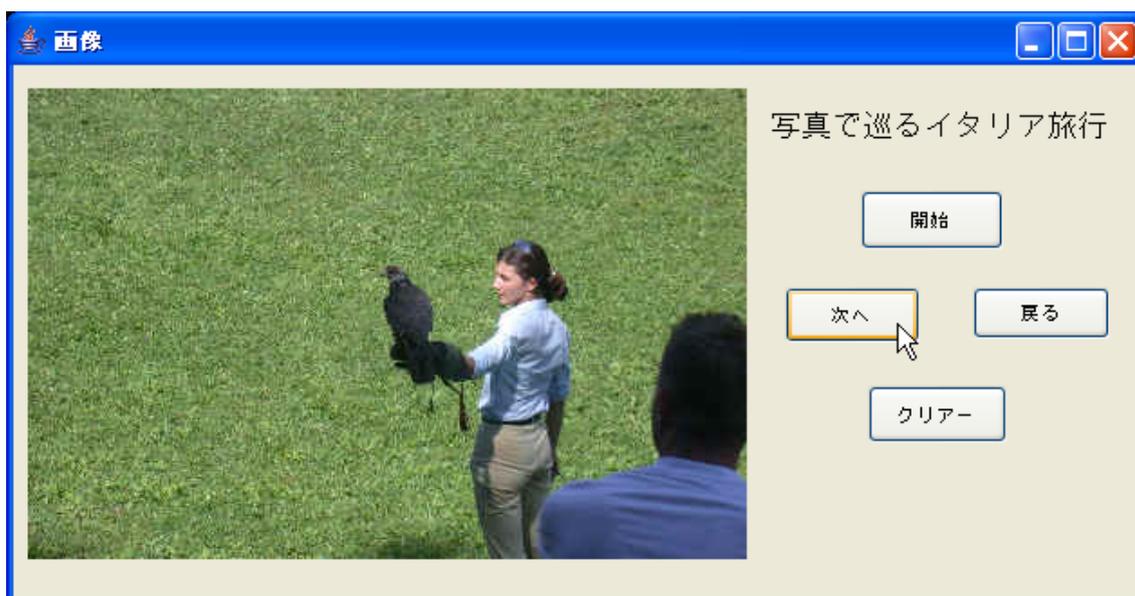


以下、[次へ] ボタンをクリックすると、順番に次の画像が表示されます。

そして5番目の画像が表示された状態で [次へ] ボタンをクリックすると・・・



下のように最初の画像に戻ります。



このように、循環的に画像を表示できるようにします。[戻る] ボタンについても同様に、循環的に一つ前の画像を表示するようにして下さい。

ヒント

- ① 何番目の画像を表示させるのか、という添え字番号 (例えば No とします) をインスタンス変数として宣言する必要があります。
- ② [次へ] ボタンをクリックしたときには、添え字番号を一つ増やしてから、そして [戻る] の場合は、一つ減らしてから (当該番号の) 画像を表示します。
- ③ ②において、 $No > 4$ になった時には、 $No = 0$ に戻す必要があります。逆に $No < 0$ になった時には、 $No = 4$ に戻す必要があります。

【応用課題 8-6-C】 キャプションの表示・消去

【応用課題 8-6-B】を改良して、次のように画像の説明（キャプション）を表示させるプログラムを作成して下さい。

前課題同様、プログラムを起動して [開始] ボタンをクリックすることで最初の画像が現れます。



ここで、[キャプション表示] ボタンをクリックすると、画像の説明が表示されます。



また、[キャプション消去] ボタンをクリックすると、キャプションは消えます。



このように、キャプションの表示/消去を全ての画像について行うようにして下さい。
各画像のキャプションは次の通りとします。

画像の順番	キャプション
1	古都シエナに伝わる鷹匠の技—美しき鷹匠
2	イタリア北部の街—バツサーノ・デル・グラッパ
3	書齋—イタリア人はアンティークな家具がお好き
4	真夏の夜のコンサート—古都ペルージャにて
5	ミラノ大聖堂—青空に映える大聖堂の尖塔

<ヒント>

- ① パネル上の Graphics クラスのオブジェクトを g とするとパネル上の座標(x,y)に文字列を表示させるには次のようにします (文字列"abc"を表示させる例です)。

```
String Moji="abc";
g.drawString(Moji,x,y);
```

ただし、この (x,y) は表示させる文字列の左下隅の座標です。

- ② 表示させる文字列のフォントを指定するには次のようにします。

```
int Fsize=14; //フォントサイズの指定
g.setFont(new Font("Dialog",Font.PLAIN,Fsize));
```

これは、フォントとして Dialog フォントを選び、通常の文字スタイルを選んだ場合です。フォントの種類と文字スタイルは次の通りです。

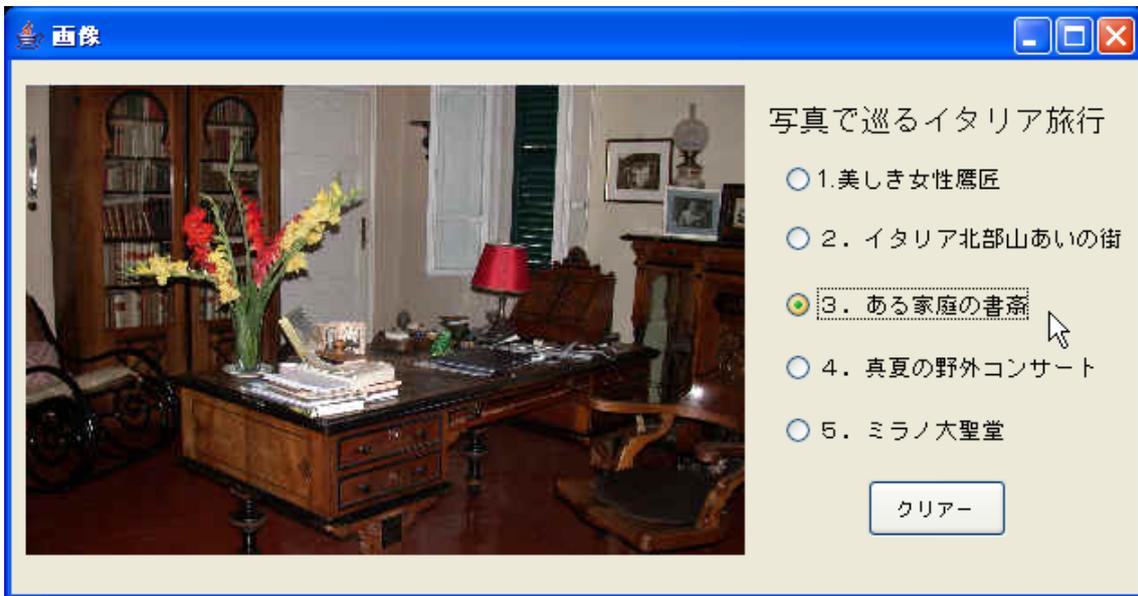
フォント名	Dialog、DialogInput、Monospaced、Serif、SansSerif
フォントスタイル	Font.PLAIN、Font.BOLD、Font.ITALIC

※ スタイルを組み合わせる場合は『 Font.BOLD|Font.ITALIC 』などとなります。

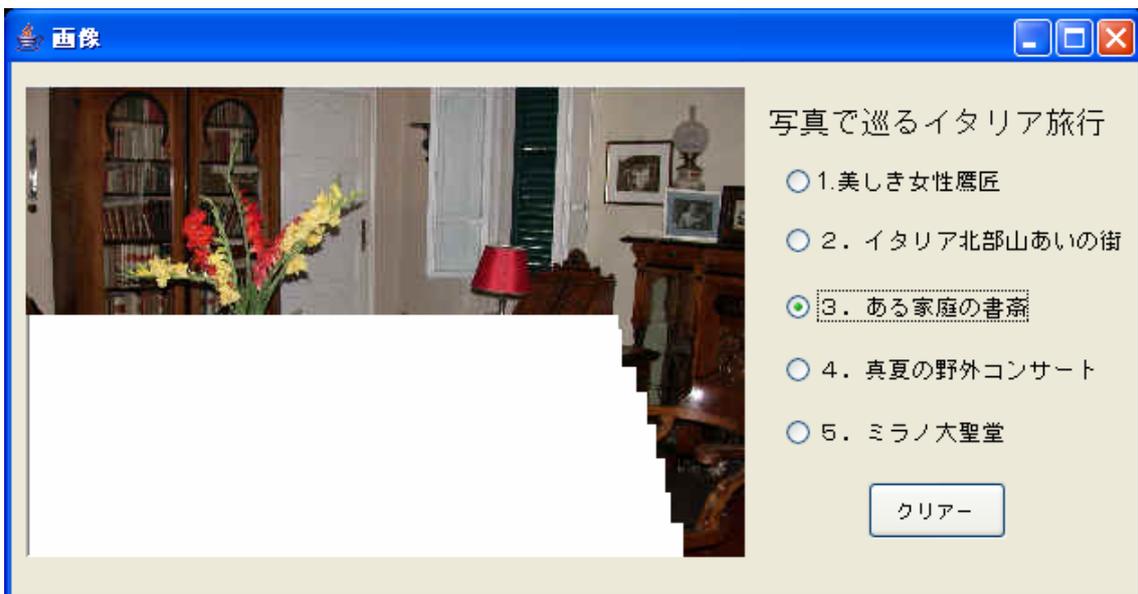
8-7 描画処理の改良—再描画処理

前節まで学んだ様に、`getGraphics()` メソッドを用いて描画したいコンポーネントの `Graphics` オブジェクトを取得し、そこに描画を行うことで、基本的にどのような CG でも作成することはできます。しかし、この方法には一つ大きな問題があります。と言うのは、(気づいた人もいるでしょうが) 一度描画した CG に、他の `Window` が重なったりすると、その部分が消えてしまうのです。

例えば【応用課題 8-6-A】で作成したプログラムで次の画像を表示させた後に…、



これが他の `Windows` と重なったりした場合は、次のようにその部分が消えてしまいます。



これは、自動的に Window が再描画されないために起こる現象です。一方、本章の冒頭 (p.192) で述べたとおり、再描画の必要がある場合は、当該コンポーネントに定義されている paint () メソッドが呼び出されます。そこで、自動的に再描画できるようにするためには、描画処理を paint () メソッド内に記述しなければなりません。しかし、最初の描画のタイミングは (プログラム開始時に自動的にではなく) これまで通り [描画] ボタンのクリック等によって、ユーザの処理で始めたいものです。以下にその方法を学習しましょう。

【準備】 ー独自のパネルコンポーネント MyPanel の定義

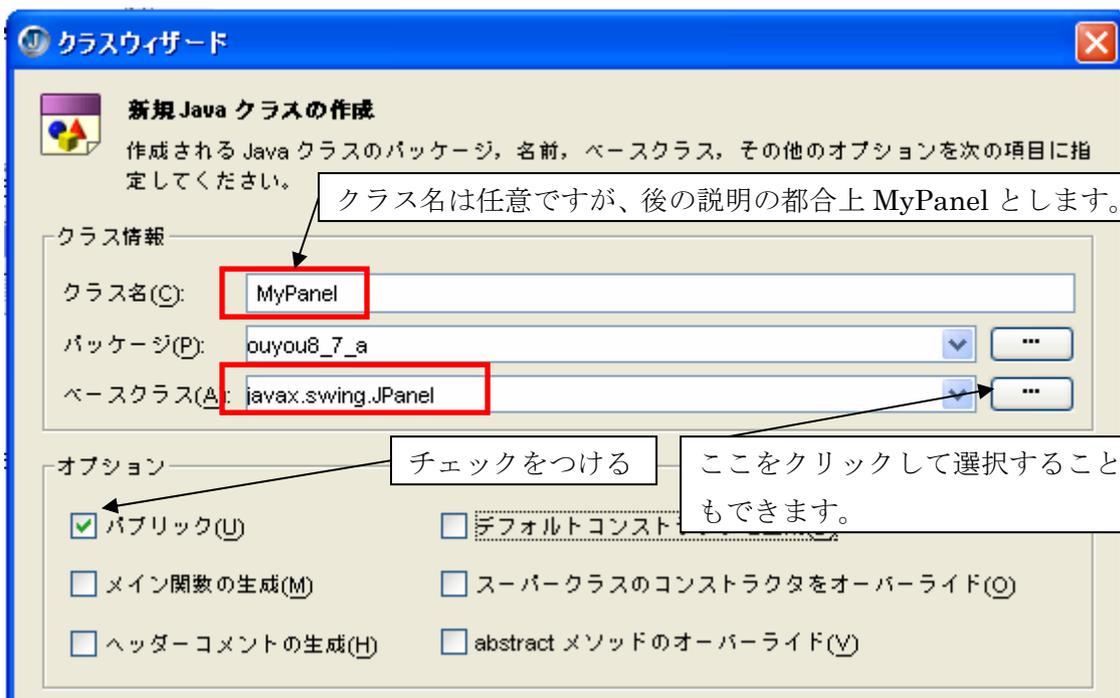
以下は、これまで通りパネルコンポーネント上に描画する場合を考えます。すると、paint () メソッドを書き換える (オーバーライドする) ためには、自分で独自のパネルクラスを定義する必要があります。その作業を以下の手順に従って進めてください。

1. 新規アプリケーションの作成

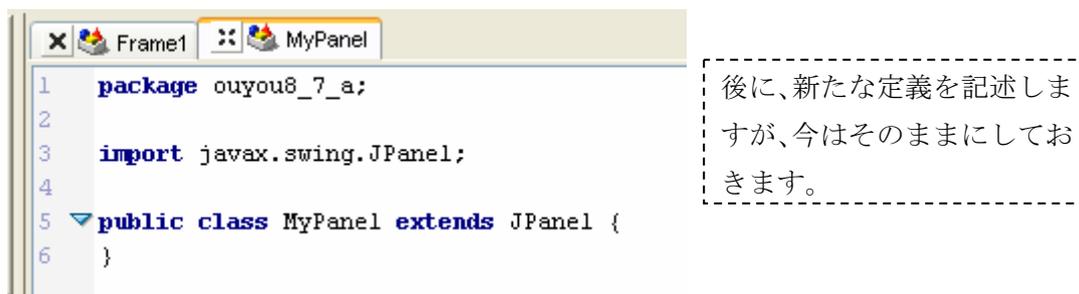
まず、いつも通りアプリケーションの新規作成を行って下さい。その際、以下の説明の都合上、プロジェクト名は「Ouyou8_7_A」としておいて下さい。

2. 新規クラス MyPanel の作成

第7章 (p.175) で行った様に、新規クラスを作成します。その際、次のように、クラス名を「MyPanel」に、ベースクラスを「javax.swing.JPanel」と記述した上で、「パブリック」欄にチェックを入れておいて下さい。



上の設定後、[OK]ボタンをクリックすると、次のクラス定義編集画面が現れます。



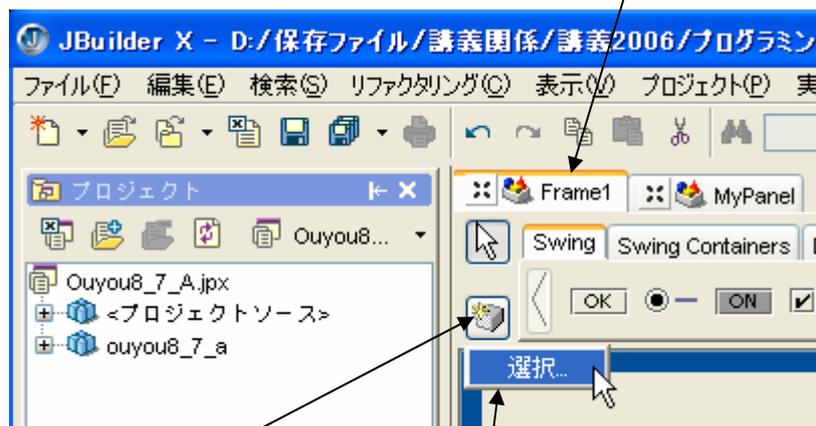
3. プロジェクトのメイク

新しいクラスをプロジェクト内で認識できるようにするために、下のように「プロジェクトのメイク」を選択します。

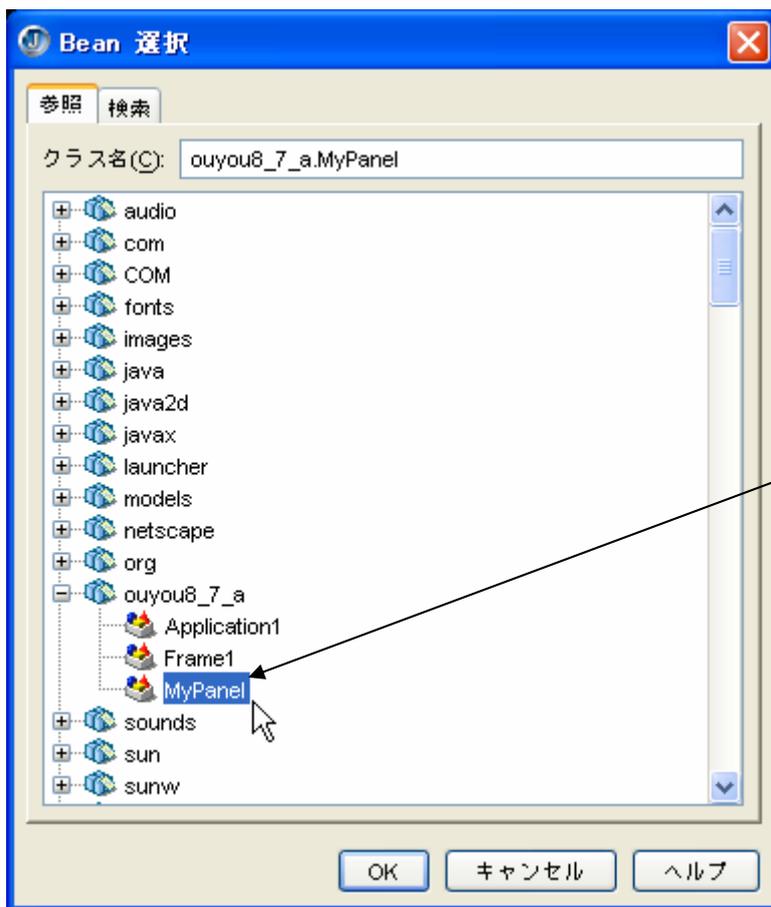


4. MyPanel コンポーネントの貼り付け

次に MyPanel コンポーネントをフレームに貼り付けます。タグをクリックして Frame1 編集画面に戻り、設計モードにして下さい。

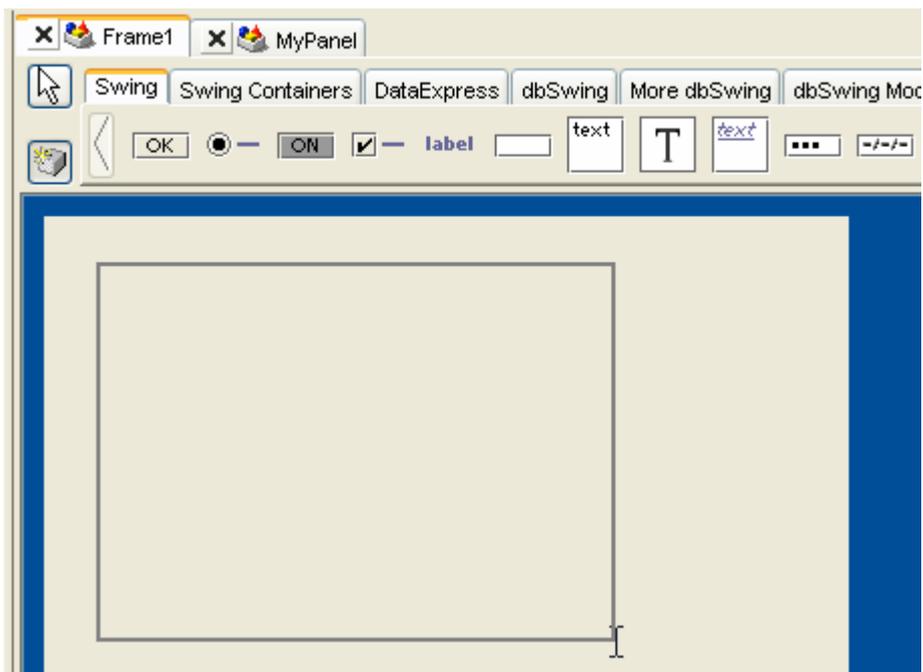


新たに定義したコンポーネントをフレームに貼り付けるためには、上のようにBean選択アイコンをクリックし、そこに現れる「選択」をクリックして下さい。

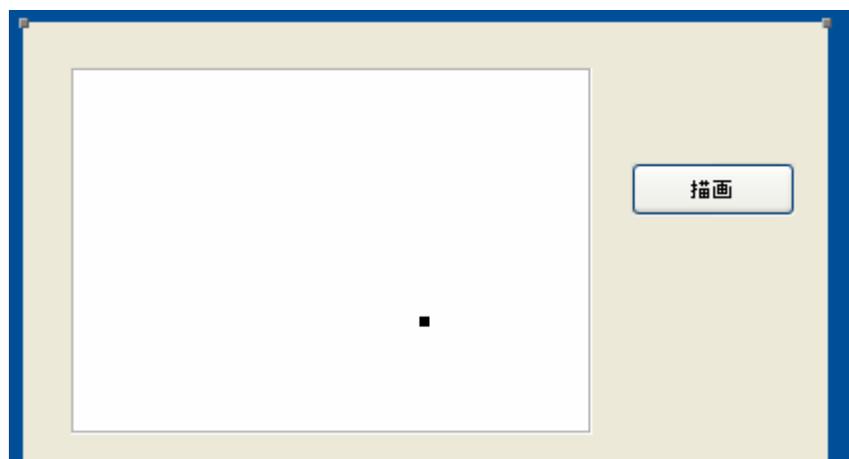


すると、左の Bean 選択画面が現れるので、今作成した MyPanel クラスをリストから選択して [OK] ボタンをクリックします。

選択後、(通常のコンポーネントと同様に) フレーム上でマウスをドラッグすることで、適当な大きさに貼り付けることができます (これまで通り、貼り付ける前に `contentPane` の `layout` プロパティを `null` にしておいて下さい)。



続いて、描画領域が分かるように **background** プロパティを白色に設定し、さらに次のような「描画」ボタンを配置しましょう。

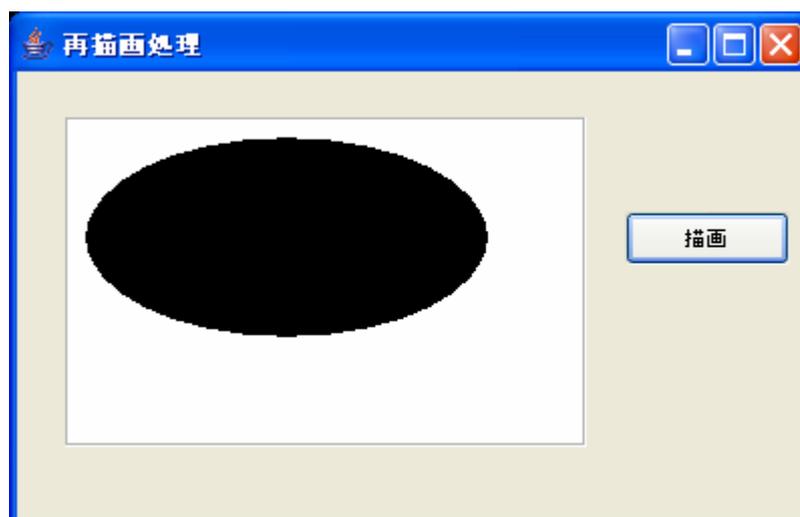


以上で、新たに定義した（現段階ではまだ何も記述していませんが）MyPanel コンポーネントをフレームに貼り付けました。同様にして、自分独自のボタンやテキストフィールド等のコンポーネントを配置することが可能です。

以上の準備が済んだら、次の【応用課題 8-7-A】に進んで下さい。

【応用課題 8-7-A】再描画処理の記述

上の MyPanel コンポーネントを使って、下のように「描画」ボタンをクリックすると、楕円を描画するプログラムを作ってみましょう。

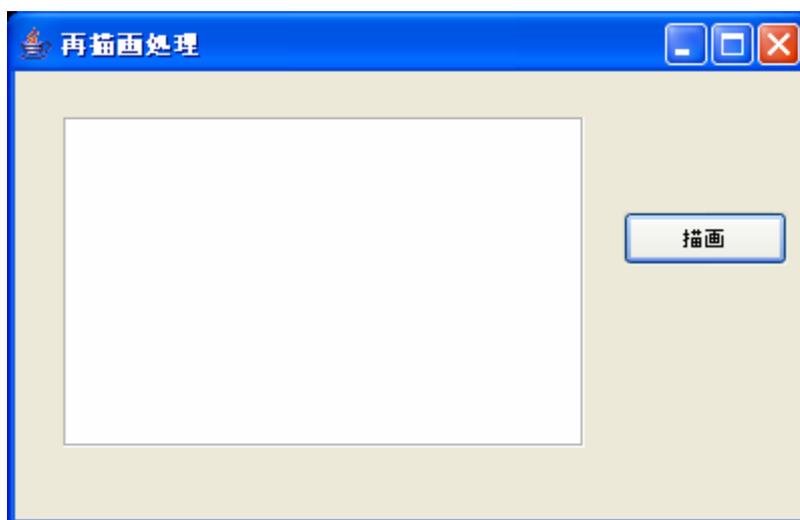


このプログラム（「描画」ボタンクリック時の処理）は難なく記述できるはずです。例えば次のように記述すれば良いでしょう。

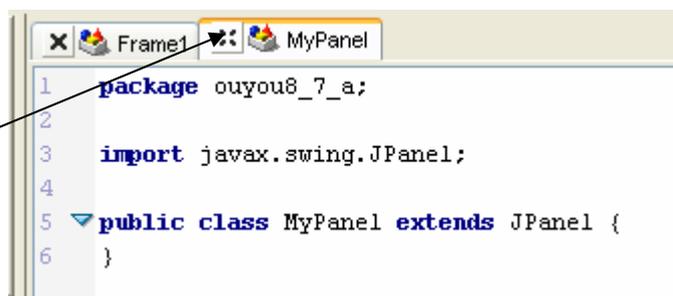
< [描画] ボタンクリック時の処理 >

```
void jButton1_actionPerformed(ActionEvent e) {  
    Graphics g=myPanel1.getGraphics();  
    g.setColor(Color.black);  
    g.fillOval(10,10,200,100);  
    g.dispose();  
}
```

しかし、これでは例えば Window を最小化した後復元すると、下のように描画した画像は消えてしまいます。これが、今改善しようとしている問題です。



そこで、再描画処理の記述に移りましょう。MyPanelクラスのソース編集画面に移って下さい。ここで、下のように下線部と枠線部を書き加えます。



```
package ouyou8_7_a;  
import javax.swing.JPanel;  
import java.awt.*; //Graphicsクラスを用いるために必要 ①  
  
public class MyPanel extends JPanel {  
    public void paintComponent(Graphics g) {  
        super.paintComponent(g);  
    } ②  
}
```

<解説>

- ① Graphics クラスは java.awt パッケージの中に含まれているので、それを用いるため import します。
- ② 描画時に呼び出されるメソッドを上書きしています。パネルなどの (Swing) コンポーネントの場合は、paint () メソッドではなく、**paintComponent ()** メソッドを用いることが推奨されています。その方が描画処理の効率がよいのです。ただしその際は、super.paintComponent (g) の様に、スーパークラスと同メソッドを呼び出す事が求められます。

さて、上は描画時に呼び出される paintComponent () メソッドを明示的に記述しただけで、プログラムの動作内容は何も変わっていません。

ここで再描画処理を考えましょう。そのアイデアは次の 2 点からなります。

1. CG を描画したときに、その内容を Image オブジェクトに保管しておく。
2. 再描画が必要な時 (paintComponent () メソッドが呼び出される時) に、その Image オブジェクトを描画する。

このアイデアに従えば、MyPanel クラスで記述すべき処理は次の 2 点です。

1. フレームオブジェクトで作成した Image オブジェクトを受け取る。
2. paintComponent () メソッドの中で当該 Image オブジェクトを描画する。

以上の考えに従って記述した MyPanel クラスの定義は次の通りです。下線部と枠線部が上の記述に書き加えたものです。意味は右側の注釈 (点線枠部) から理解できるでしょう。

```
public class MyPanel extends JPanel {  
    private Image Img=null; //Image オブジェクトの宣言  
  
    public void paintComponent (Graphics g) {  
        super.paintComponent (g);  
        if (Img!=null) {  
            g.drawImage (Img, 0, 0, this);  
        }  
    }  
  
    public void setImage (Image img) {  
        Img=img;  
    }  
}
```

作成された Image オブジェクトの表示 (まだ作成されていない時つまり null の場合は表示しない事に注意)。

フレームオブジェクトで作成した Image オブジェクトをインスタンス変数 Img として受け取る。

上を記述したら、Frame1 の編集画面に戻り、[描画] ボタンの処理に下のように書き加えて下さい。枠内が、以前 (p.222) の記述に加えた部分です。

```
void jButton1_actionPerformed(ActionEvent e) {
    Graphics g=myPanel1.getGraphics();
    g.setColor(Color.black);
    g.fillOval(10,10,200,100);
    g.dispose();

    int w=myPanel1.getWidth(); //パネル幅の取得
    int h=myPanel1.getHeight(); //パネル高さの取得
    Image img=this.createImage(w,h); //Image オブジェクトの生成 ①
    Graphics g2=img.getGraphics(); //Graphics オブジェクトの取得 ②
    g2.setColor(Color.white);
    g2.fillRect(0,0,w,h);
    g2.setColor(Color.black);
    g2.fillOval(10,10,200,100);
    myPanel1.setImage(img); //画面を Image オブジェクトとして設定 ④
    g2.dispose();
}
```

③ 背景を白で塗り、楕円を描くという処理を、コピー用の Image オブジェクトにも同様に施している。

<解説>

- ① createImage(w,h) メソッドは Image オブジェクトを生成するメソッドで、フレームクラスに備わっています。これにより、幅w、高さhの大きさを持つ Image オブジェクトを生成できます。
- ② Image オブジェクトから getGraphics()メソッドにより Graphics オブジェクトを取得します。
- ③ この部分の処理により、Image オブジェクト上にも、パネルと同様の描画処理がなされています。
- ④ 描画後、Image オブジェクトを setImage()メソッドにより MyPanel オブジェクトに渡すことで、再描画時に、当該 Image オブジェクト (の描画内容) が表示されるようになります。

作成したら実行して下さい。そして、一度描画すると最小化等により一度 Window が画面から消えても、復元時に再描画がなされる (画像が復元される) ことを確認して下さい。

ここで説明した方法により、本章で作成した CG は全て (必要な時に) 再描画がなされるように改良する事ができます。