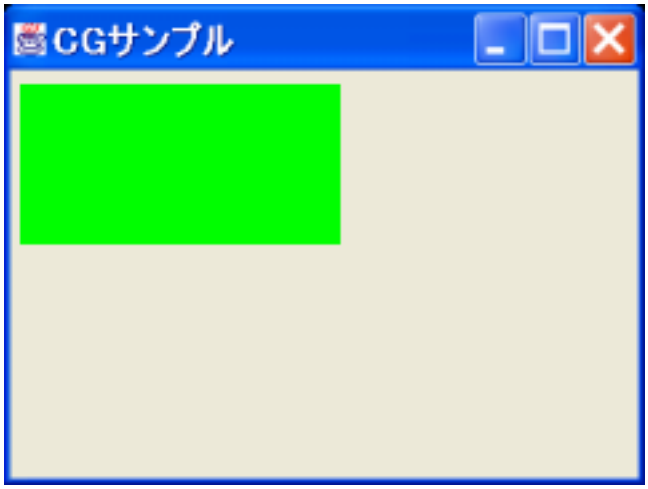
２００３年度プログラミング応用編ー課題III

HP に課題プログラムのクラスファイルを掲載しています。このクラスファイルをダウンロードすると、該当プログラムを実行し動作結果を確認できます。やり方は「クラスファイルからの実行の仕方」を参照して下さい。課題名横の（　）内に記してあるのがクラスフォルダの名前です。なお、課題名の右端に記してある数字は、解いた場合に成績に加点される得点です。

今回は、Java を用いた CG（コンピュータグラフィックス）の描き方（作成の仕方）を学習します。Java 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。

CG の描き方①ー標準的な方法 (paint メソッドを使用)

では、次のような CG を描いてみましょう。これは、実行すると、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、コンポーネントに砂割っている paint メソッドに、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。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、再描画する際に自動的に呼び出されます。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自動的に（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。

では、適当なプロジェクトを新規作成し、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の下に、次のように 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。

```
//フレームのビルド
public Frame1() {
  enableEvents(AWTEvent.WINDOW_EVENT_MASK);
  try {
    jbInit();
  }
  catch(Exception e) {
    e.printStackTrace();
  }
}
```

新たに記述した部分

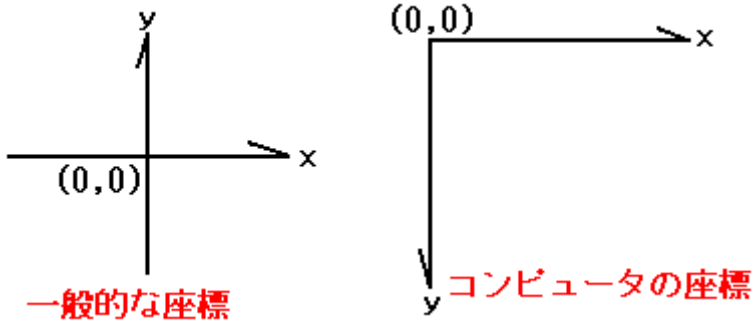
```
public void paint(Graphics g) {
  g.setColor(Color.green);
  g.fillRect(10,50,200,100);
}
```

＜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 と言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。

③ fillRect(x,y,w,h)メソッドは、点（x1,y1）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。



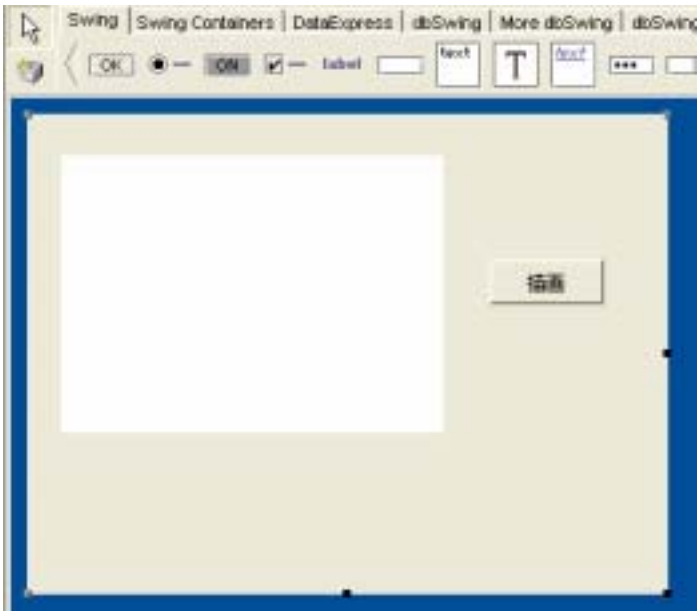
CG の世界では、左上を原点(0,0,)とし、y 座標が下向きに伸びていることに注意して下さい。

作成したら、プログラムを実行し動作を確認して下さい。

CG の描き方②ーgetGraphics()メソッドを用いる方法（より融通が利く）

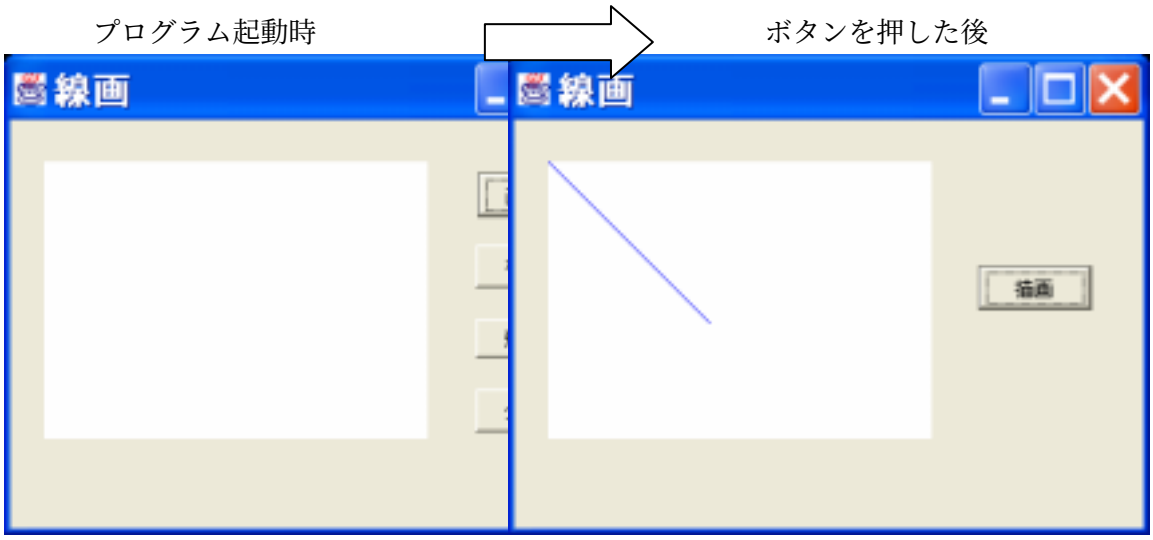
上の paint メソッドを用いる方法は簡単なのですが、ボタンを押したときに CG 描画を開始させるなど、CG 作成の動作制御を行う際には少し不便です。そこで、以下では、対象とするコンポーネントの Graphics オブジェクトを直接取得するメソッド `getGraphics()`を用いる方法を用いることにします。

用い方はやはり簡単です。新しいプログジェクトを作成し、下のように、パネルコンポーネントを貼り付け、その background プロパティを白色にして下さい。そして［描画］ボタンを貼り付けます。



今、プログラムを起動し、［描画］ボタンを押すと下のような直線を（青色で）描くプログラムを作成しましょう。ボタンのイベントハンドラは次ページの通りです。

プログラム起動時　　ボタンを押した後



```
void jButtonDraw_actionPerformed(ActionEvent e) {
  Graphics g=jPanel1.getGraphics(); //Graphics オブジェクトの取得
  g.setColor(Color.blue);
  g.drawLine(0,0,100,100);
}
```

＜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。

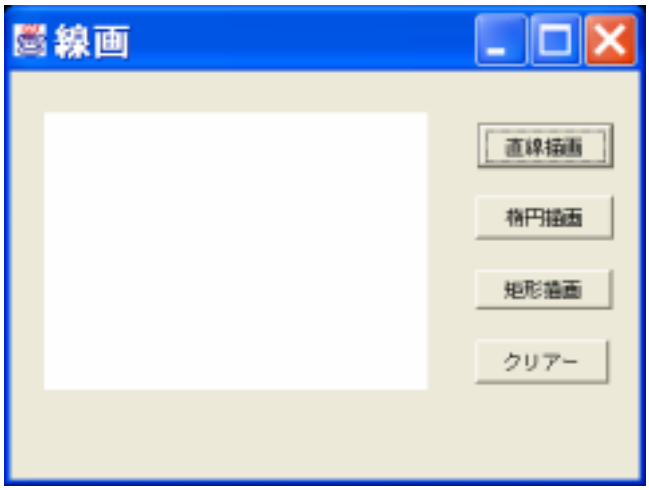
② ２行目で使用する色を青色にしています。

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。

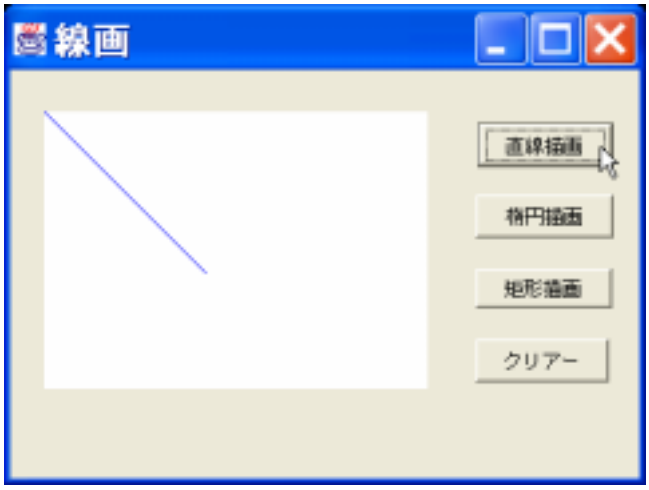
④ 今の場合、Graphics オブジェクト g が（フレームではなく）パネルコンポーネントのオブジェクトなので、CG 作成はパネル上で行うことに注意して下さい。このように getGraphics()メソッドを用いれば、ボタンやラベルの上など、様々なコンポーネントの Graphics オブジェクトを取得でき、したがって当該コンポーネント上での CG 作成を容易に行えます。

色々な図形の描画練習

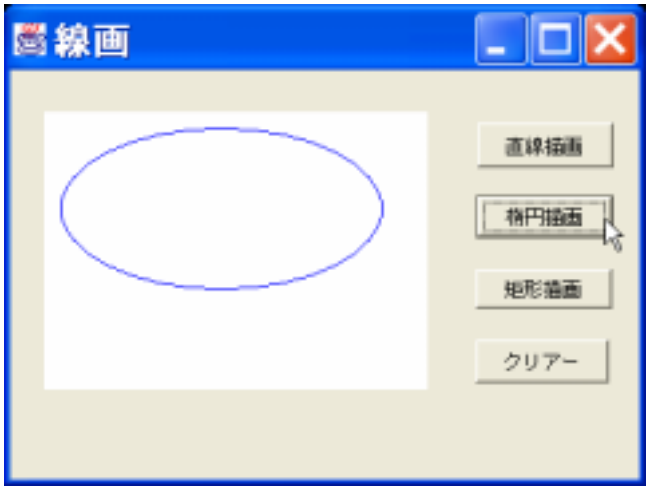
それでは、代表的な図形を描く練習をしてみましょう。作成するのは次のようなプログラムです。



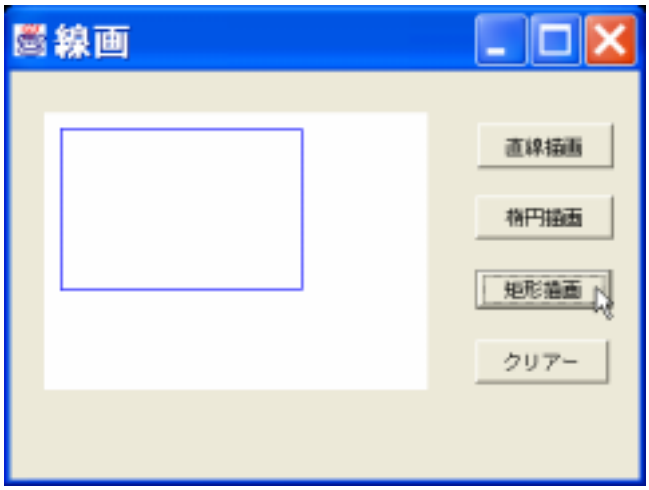
起動すると次のような画面が現れる。



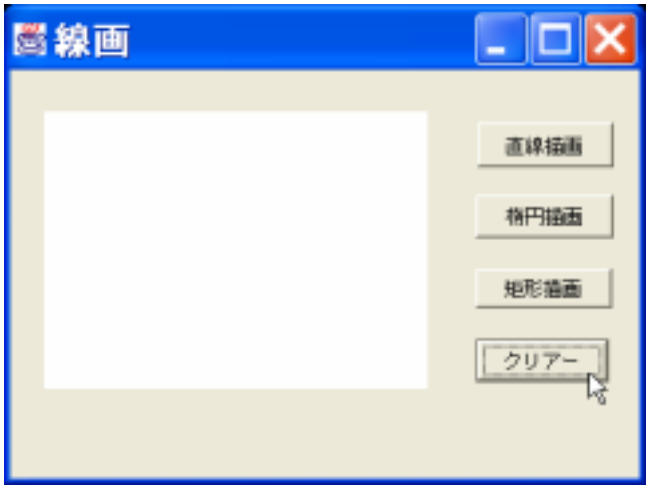
［直線描画］ボタンをクリックすると直線を描く。



［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。

＜［直線描画］ボタン＞

```
void jButtonLine_actionPerformed(ActionEvent e) {
  Graphics g=jPanel1.getGraphics();
  g.setColor(Color.blue);
  g.drawLine(0,0,100,100);
}
```

＜［楕円描画］ボタン＞

```
void jButtonOval_actionPerformed(ActionEvent e) {
  Graphics g=jPanel1.getGraphics();
  g.setColor(Color.blue);
  g.drawOval(10,10,200,100);
}
```

＜［矩形描画］ボタン＞

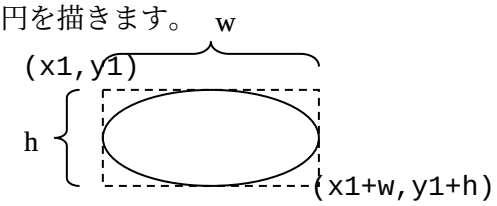
```
void jButtonRect_actionPerformed(ActionEvent e) {
  Graphics g=jPanel1.getGraphics();
  g.setColor(Color.blue);
  g.drawRect(10,10,150,100);
}
```

＜［クリアー］ボタン＞

```
void jButtonClear_actionPerformed(ActionEvent e) {
   int XMax=jPanel1.getWidth(); //パネルの幅の取得
   int YMax=jPanel1.getHeight(); //パネルの高さの取得
   Graphics g=jPanel1.getGraphics();
   g.setColor(Color.white);
   g.fillRect(0,0,XMax,YMax);
 }
```

＜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 ［クリアー］ボタンのイベントハンドラの処理内容は、枠内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。

課題3－1　色々な図形の描画　　2

次のようなプログラムを作成して下さい。

起動すると左のような画面が現れます。

「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。

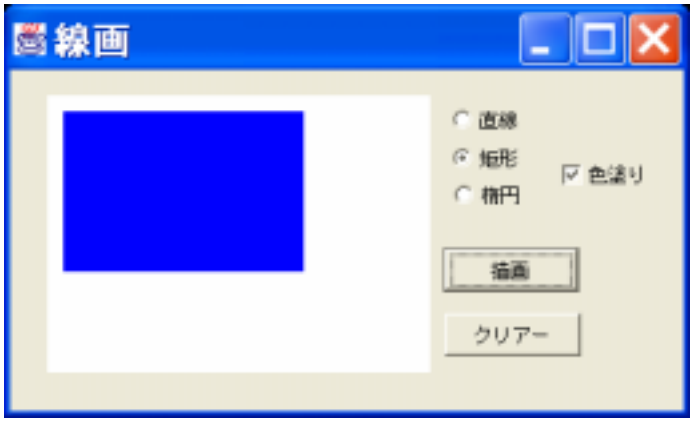
［描画］ボタンのイベントハンドラを次のように記述しました。

```
void jButtonDraw_actionPerformed(ActionEvent e) {
  Graphics g=jPanel1.getGraphics();
  if(jRadioButtonLine.isSelected()) {
    LineDraw(g);  //線を描くメソッド
  }
  else if(jRadioButtonRect.isSelected()) {
    RectDraw(g);  //四角形を描くメソッド
  }
  else {
    OvalDraw(g);  //楕円を描くメソッド
  }
}
```

メソッド LineDraw(g)、RectDraw(g)、OvalDraw(g)を定義して下さい。

課題3－2　色塗りの追加(zukeidraw)　1

課題４－１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加えて下さい。例えば、下のように「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。



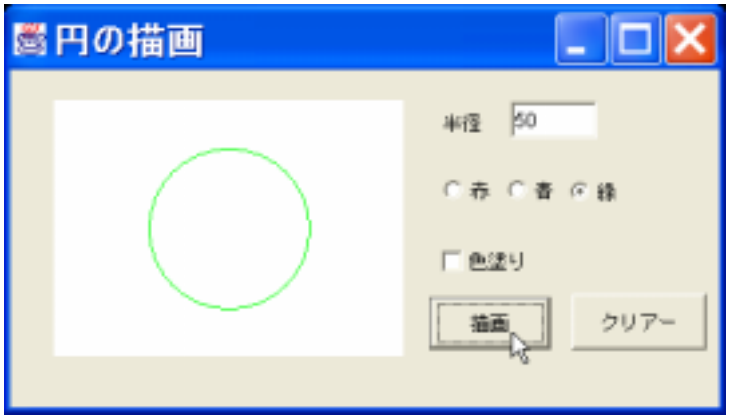
※　直線に関しては、色塗り欄は無効（チェックの有無は関係ない）です。

課題3－3　任意の円の描画(circle)　2

次のように、指定した半径と色に応じて円を描くプログラムを作成して下さい。

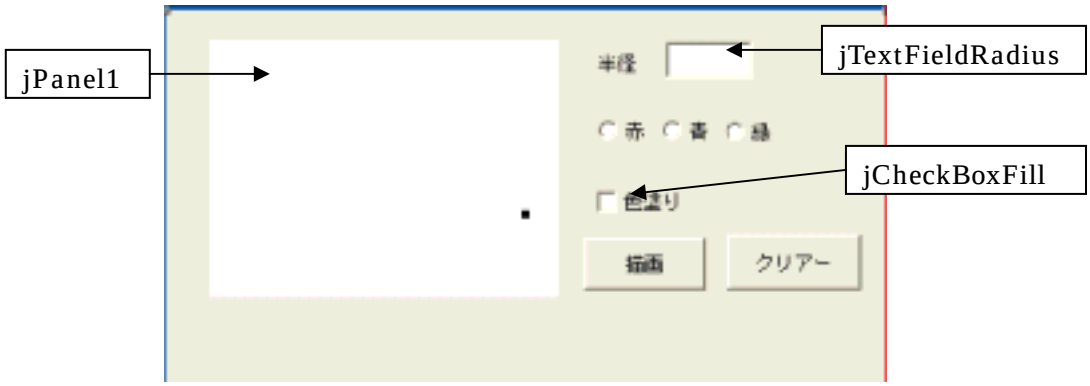
起動すると左のような画面が現れます。

半径欄に値を代入し、色を選んで色塗り欄をチェックすると、指定色で色を塗りつぶした円を描きます。



色塗り欄がチェックされていない場合は、円を描画します。

主立ったコンポーネントの name プロパティを次の通りとします。



［描画］ボタンクリック時のイベントハンドラを次のように作成しました。空欄を埋めてプログラムを完成させて下さい。また、点線枠内のメソッド Circle()は指定した条件で円を描く（独自の）メソッドです。この定義を完成させて下さい。

```
void jButtonDraw_actionPerformed(ActionEvent e) {
  Graphics g=jPanel1.getGraphics();
  int r=Integer.parseInt(jTextFieldRadius.getText()); //半径ｒの定義
  Color cl;  //Color クラスの変数（オブジェクト）の宣言
  boolean Fill=jCheckBoxFill.isSelected(); //色塗りの有無判定用論理型変数
  int xc=jPanel1.getWidth()/2;  //円の中心のｘ座標
  int yc=jPanel1.getHeight()/2; //円の中心のｙ座標
  if(                                    ) {  //赤が選択された場合
    cl=Color.red;  //色オブジェクト cl に赤を代入
  }
  else if (                                    ) {  //青が選択された場合
    cl=Color.blue;  //色オブジェクト cl に青を代入
  }
  else {  //その他（今の場合緑）が選択された場合
    cl=Color.green;  //色オブジェクト cl に緑を代入
  }
  Circle(g,xc,yc,r,cl,Fill);  //円を描くメソッドを呼び出す
}
```

課題3－4　多角形の描画(polygon)　2

任意の多角形を描く Polygon メソッドの用い方を学習しましょう。例として、次のように、三角形を描くプログラムを考えます。

プログラムを起動し、［描画］尾胆をクリックすると、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=jPanel1.getGraphics();
  int xc=jPanel1.getWidth()/2;  //パネルのｘ座標の中点を求める
  int yc=jPanel1.getHeight()/2; //パネルのｙ座標の中点を求める
  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);
  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
}
```

＜解説＞

① 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。

② 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、下の説明を参照して下さい。

③ 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[n-1],y[n-1])、そして再び点（x[0],y[0]）までを順に直線で結びます。

④ 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。

＜配列＞

配列とは x[1]、x[2]、･･･などの様に、変数の値を[]内の数字（**添え字**と呼びます）で指定できる変数のことです。点１～点 N までのｘ座標を、x_1、x_2、･･･、x_Nなどと表しますが、これをプログラムでは、x[1]、x[2]、･･･x[N]と表します。これが配列という訳です。配列を宣言する場合は、その大きさ（個数）が N の場合、以下のように記述します。

整数型の場合　→　`int x[]=new int[N];`

実数型の場合　→　`double x[]=new double[N];`

文字列型の場合　→　`String x[]=new String[N];`

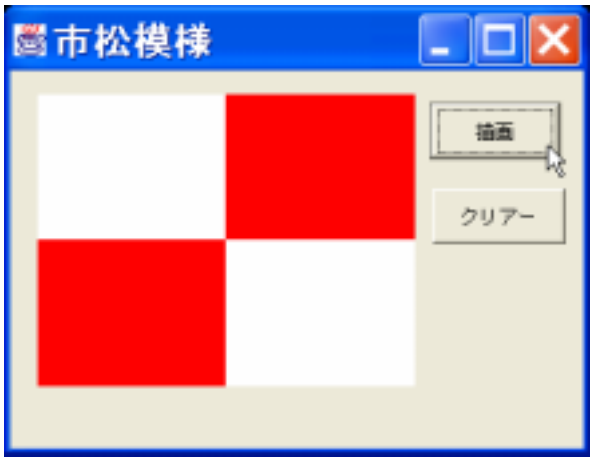
なお、配列の添え字は**0（番目）から始まる**ことに注意して下さい。

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。

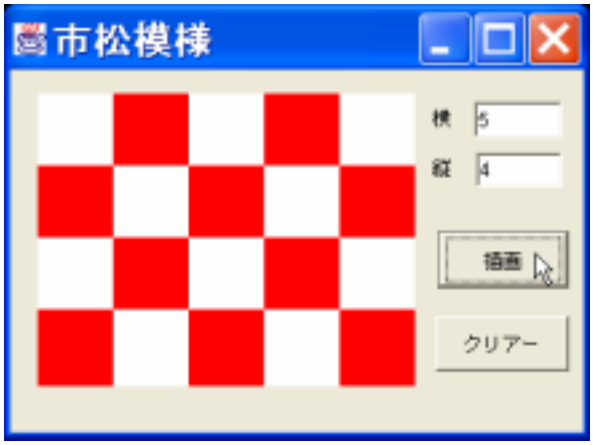
課題3－5　市松模様　　1

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。右の例では、白色と赤色で塗り分けています。



課題3－6　市松模様2(ichimatsu)　3

上のプログラムを発展させ、次のように、任意の縦・横の数を入れて［描画］ボタンをクリックすると、指定した縦・横数の市松模様を描くプログラムを作成して下さい。



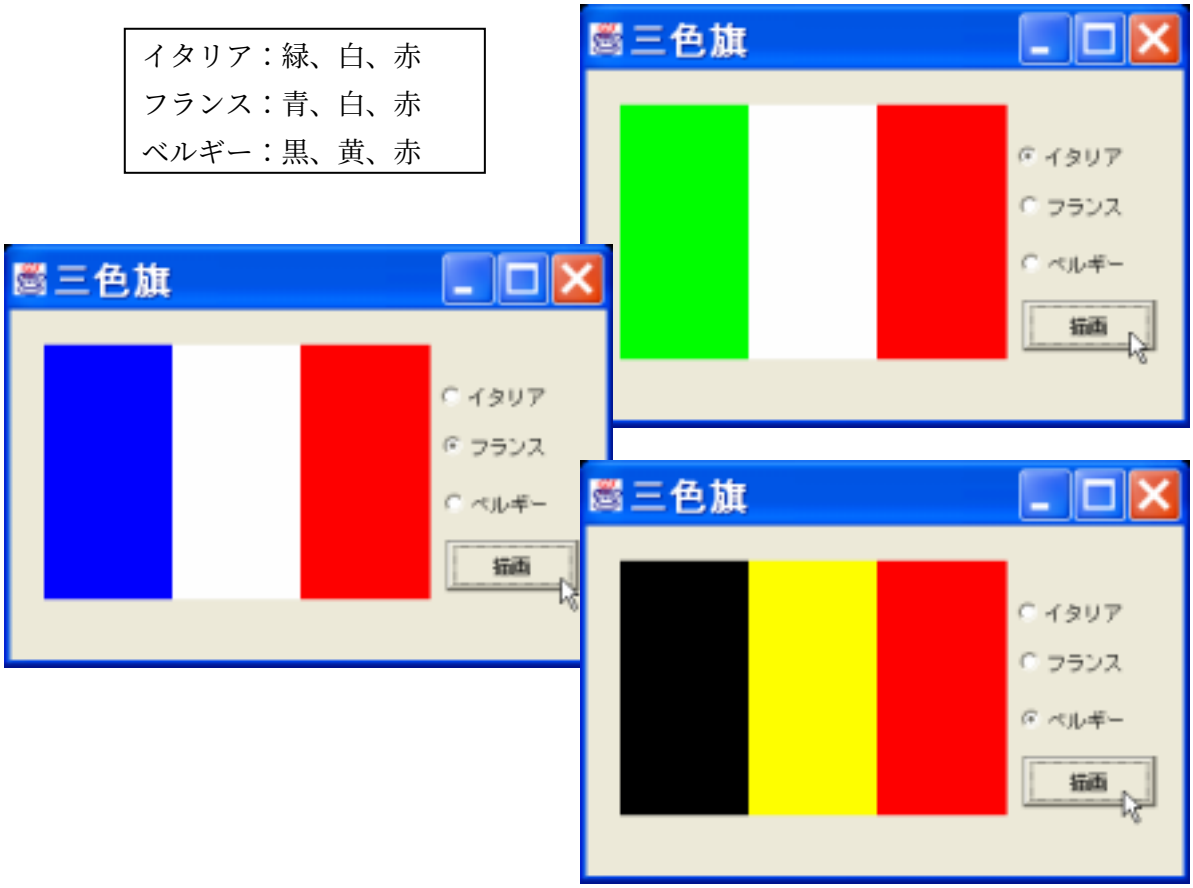
課題3－7　ダイヤを描く(daiya)　2

次のように、縦・横の数を入力して［描画］ボタンをクリックすると、その数に応じてダイヤを描くプログラムを作成して下さい。

課題3－8　三色旗(flag)　　2

下のように、国名を選択して［描画］ボタンをクリックすると、該当する三色旗を描くプログラムを作成して下さい。

イタリア：緑、白、赤
フランス：青、白、赤
ベルギー：黒、黄、赤



課題3－9　　点画ーカオスのアトラクタ(chaos)　　3

drawLine(x1,y1,x2,y2)メソッドは、２点の座標を同じにするとその点に指定色で点を打つ命令になります。そこで、これを使って、点画を描いていましょう。

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

$$X_{n+1}=Y_n-0.97*X_n+5/(1+X_n^2)-5, \qquad Y_{n+1}=-0.995*X_n$$

最初の点を（X_1,Y_1）=(1,0)としてこの点列を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

点の数を 30000 にして描画したところ

ヒント

① 上の（X_{n+1}, Y_{n+1}）は小さな数になるので見にくくなります。そこで実際には（$6*X_{n+1}$, $6*Y_{n+1}$）と 6 倍拡大して表示しています。

② 座標の原点は Image コンポーネントの中心となるようにしています。

③ パネルコンポーネントの幅と高さは約 200 程度にしています。

④ （X_{n+1}, Y_{n+1}）が求まるたびにその点（実際には 6 倍拡大した座標）に点を打てば良いのです。その際、（X_{n+1}, Y_{n+1}）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。