Chapter 6. グラフィックスと繰返し

６－１．グラフィックスの描画

６－１－１．AWTパッケージとグラフィックキャンパスクラス

setColor, drawStringといったメソッドは、前にも紹介しましたが、これは、JavaのAWT（Abstract Window Toolkit）と呼ばれるクラスライブラリ（パッケージ）の中のグラフィックス（Graphics）クラスの中に定義されているものです。このクラスには、いろいろな描画を行なうためのメソッドが他にも用意されています。ここで、それらのうちの幾つかを紹介します。

まず、このAWTのクラスを使うときには、次のように、import文を使って、クラスを用いると指定を予めしておく必要があります。

```
import  java.awt.* ;                    // これからAWTのクラスを使う準備をせよ
```

この指定の「＊」の部分は、すべてのクラスを使うという意味です。例えば、Graphicsクラスしか使用しないのであれば、次のように書くこともできますが、どのクラスを利用するか予め見当がつくことが少ないので、たいていは「＊」で指定することの方が多いようです。

```
import  java.awt.Graphics;              // AWTパッケージのGraphicsクラスを使う準備をせよ
```

６－１－２．図形を描くメソッド

図形を描くメソッドを紹介していきます。これらのメソッドは、paintメソッドでパラメータとして受け取ったグラフィックスオブジェクトを描画領域の対象としていますので注意してください。例では、gという変数を用いて、このグラフィックスオブジェクトのことを指し示すこととします。説明では、ある点のx座標とy座標の組を、(x, y)という形で表現します。座標は、ドット単位（整数あるいは整数の式）で記述します。

★線を描くメソッド

▼drawLine(始点のx座標 , 始点のy座標 , 終点のx座標 , 終点のy座標) ;
　　→始点から終点まで線を引きます。

始点と終点は、どちらをどちらにしても同じです。ですから、次の２つのメソッド呼出しは共に同じ直接を引くことになります。

```
g.drawLine( 20, 30,  200, 180 );
g.drawLine( 200, 180, 20, 30 );
```

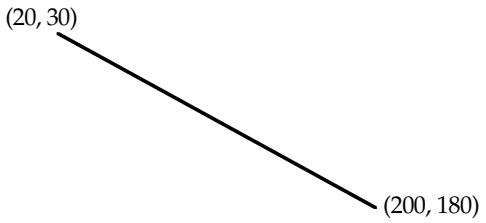


図6-1 線の描画例

★四角形を描くメソッド

▼drawRect(始点のx座標 , 始点のy座標 , 幅 , 高さ) ;
　　→始点から、幅・高さで指定された四角形の枠を描きます。

始点は、左上の角のことです。drawLineとdrawRectでは、３番目と４番目のパラメータの意味が違うことに注意してください。例えば、以下の２つのメソッド呼出しでは右下の端が違う位置になります。

```
g.drawLine( 10, 10, 100, 100 );        // (10, 10 ) → (100, 100 )
g.drawRect( 10, 10, 100, 100 );        // (10, 10 ) → (110, 110 )
```

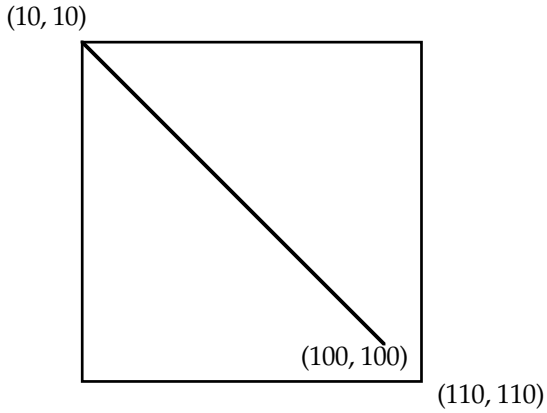


図6-2 四角形と線の描画の差

★楕円や正円を描くメソッド

▼drawOval(始点のx座標 , 始点のy座標 ， 外接する四角形の幅 ， 外接する四角形の高さ);
→楕円や正円を描きます

円から見て外接する四角形の幅と高さが等しければ、正円になります。例えば、次のメソッド呼出しでは、幅80ドット、高さ50ドットの四角形に内接する円を描きます。これが、例えば、幅80ドット、高さ80ドットの場合には、正円になります。

```
g.drawOval( 10, 10,  80, 50 );
```

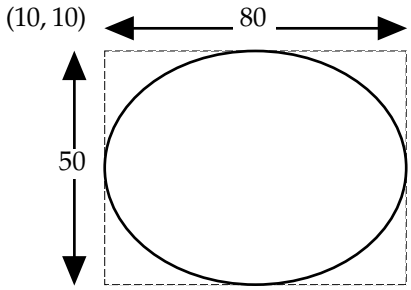


図6-3 楕円の描画例

★弧を描くメソッド

▼drawArc(始点のx座標 , 始点のy座標 ， 外接四角形の幅 ， 外接四角形の高さ,
開始する角度 , 角度差);
→弧を描きます。

開始角度や角度差は、通常の角度指定（degree）で行ないます。整数あるいは整数の式で記述します。-360°～360°の範囲で指定します。極座標と同じように反時計回りになっています。開始角度の0°は、右の水平方向になっています。

```
g.drawArc(  20, 20,  70, 60, 135, 90 );
```

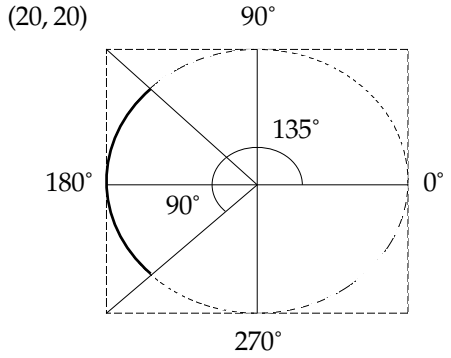


図6-4 弧の角度と描画例

★丸みを帯びた四角形を描画するメソッド

▼drawRoundRect(始点のx座標 , 始点のy座標 , 幅 , 高さ , 丸みの幅 , 丸みの高さ);
→始点から、幅・高さで指定された丸みを持った四角形を描く。

丸みの幅と高さも整数式で指定します。もし、幅や高さに対して、丸みの幅や高さがその半分の大きさになってしまうと、俵型を表示することになるでしょう。

```
g.drawRoundRect( 100, 100, 50, 40, 10, 10 );
```

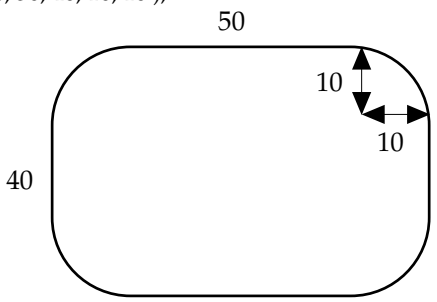


図6-5 丸みを帯びた四角形の描画例

★３次元的な四角形を描画するメソッド

▼draw3DRect(始点のx座標 , 始点のy座標 , 幅 , 高さ , 凹凸の指定);
→３次元的な四角形を描画します。

凹凸の指定は、論理値で行います。**true**だと、飛び出して見えます。**false**だと窪んで見えます。ただし、色を灰色系（Color.grayなど）にしないとそれっぽく見えないので注意が必要です。

```
g.setColor( Color.gray );
g.draw3DRect( 10, 10, 100, 100, true );
g.draw3DRect( 50, 50, 20, 20, false );
```

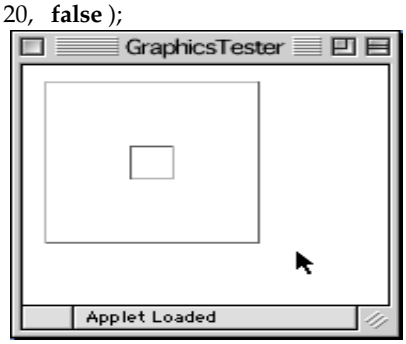


図6-6 ３次元的四角形の描画例

６－１－３．塗りつぶしで図形を描くメソッド

枠を書くメソッドに対応して、その内部を塗りつぶすメソッドも次のようなものが用意されています。

```
fillRect( 始点のx座標 , 始点のy座標 , 幅 , 高さ );
fillOval( 始点のx座標 , 始点のy座標 , 外接四角形の幅 , 外接四角形の高さ );
fillArc( 始点のx座標 , 始点のy座標 , 幅 , 高さ, 開始角度 , 角度差 );
fillRoundRect( 始点のx座標 , 始点のy座標 , 幅 , 高さ , 丸みの幅 , 丸みの高さ );
fill3DRect( 始点のx座標 ,始点のy座標 , 幅 , 高さ , 凹凸の指定 );
```

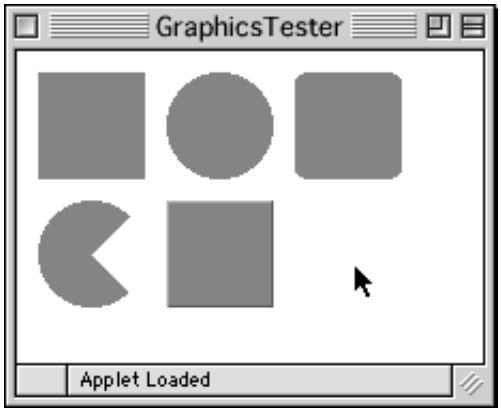


図6-7 塗りつぶしの描画例

６－１－４．彩色と描画

描画や塗りつぶしのときの色を変えるためには、予め色を設定しておくことが必要です。setColorメソッドで色を設定した場合は、その後の描画はすべてその色で行なわれます。別の色にしたければ、またsetColorメソッドを使って色を設定する必要があります。なお、アプレットが起動されたときの最初の描画色は黒色に設定されています。

```
import java.awt.* ;
import java.applet.* ;

public class ColoredDrawing extends Applet {
        public void paint( Graphics g ) {
                g.setColor( Color.red );            // 以降は赤色で描画される
                g.drawLine( 0, 0, 100, 100 );
                g.drawRect( 100, 0, 100, 100 );
                g.setColor( Color.green );          // 以降は緑色で描画される
                g.drawLine( 250, 0, 350, 100 );
                g.drawRect( 350, 0, 100, 100 );
        }
}
```

６－１－５．コピーエリアなど

描画以外にもいろいろなメソッドが用意されています。

▼clearRect(始点のx座標 , 始点のy座標 , 幅 , 高さ);
　　→四角形の領域を背景色で塗りつぶします。

▼copyArea(始点のx座標 , 始点のy座標 , 幅 , 高さ , 横の移動量 , 縦の移動量) ;
→四角形の領域に書かれている内容を縦横に移動した後コピーして表示します。

copyAreaは、移動した後の四角形の領域が元の四角形の領域と重なるときは、移動後の方が優先されますので、注意してください。なお、移動量はマイナスの値も取ることができます。以下のサンプルプログラムを実行してみたり、移動量のパラメータの値を変えたりして、確かめてみてください。

```
import  java.awt.* ;
import  java.applet.* ;

public class GraphicsTester  extends  Applet {
        public  void  paint( Graphics  g ) {
                g.fillRect( 10, 10, 80, 80 );
                g.setColor( Color.red );
                g.fillOval( 50, 50, 80, 80 );
                g.copyArea( 10, 10, 120, 120, 150, 20 );  //  横方向150, 縦方向20の位置にコピー
        }
}
```

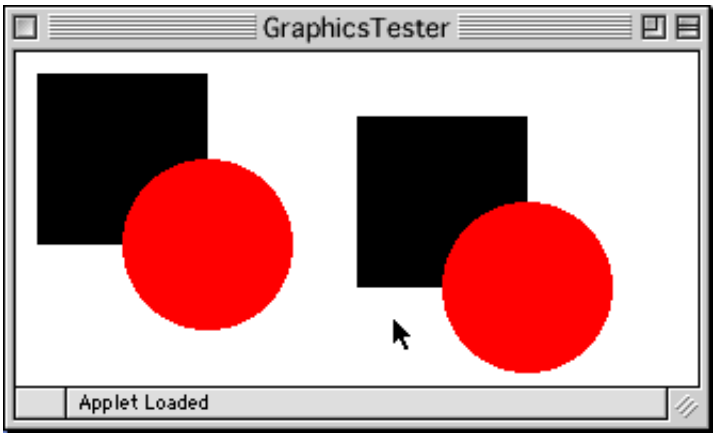


図6-8 コピーエリアの実行例

６－２．グラフィックスと繰返し

グラフィックスで図形を表示させるメソッドには、drawLineの他にdrawRect、drawOvalあるいはdrawArcなど、中を塗りつぶすメソッドも含めるとかなり多数のメソッドを取り上げてきました。ここでは、その中で一番基礎的なdrawLineを使って、繰返しと組み合わせたときの効果を復習します。また、メソッド呼出しのためのパラメータを、どのように設計しなければならないかということも、drawLineを使って学んでいきましょう。

６－２－１．drawLineとwhile文による１次関数グラフ

for文とdrawLineを使って、様々な図が掛けることに注目しましょう。まず、drawLineをパラメータの指定の仕方を復習してみましょう。

drawLine(始点のx座標 , 始点のy座標 , 終点のx座標 , 終点のy座標) ;

縦線と横線についてパラメータがどうなっているかを考えてみると、次のようになります。

縦線	始点のx座標と終点のx座標が等しい
横線	始点のy座標と終点のy座標が等しい

たとえば、横線を使って平行四辺形を書くことを企図してみましょう。このときは、始点も終点もy座標はfor

文で使うループ変数をy座標に当てはめてみます。x座標はループ変数を使って、幅を持たせてみます。

```
public void paint (Graphics g ) {
        int  i=1;
        while  ( i<=100 ) {
                g.drawLine( 100-i,  i,  200-i,  i );
                i++;
        }
}
```

図6-9　横線で描画した平行四辺形

始点と終点のx座標を変えましたが、始点は固定し、終点のx座標だけ変えてみると次のような図になります。

```
public void paint (Graphics g ) {
        int  i=1;
        while  ( i<=100 ) {
                g.drawLine( 100,  i,  100+i,  i );
                i++;
        }
}
```

図6-10　始点のx座標を固定した場合

同じことは縦線でy座標を変えて行くような場合にも適用できます。

６－２－２．while文と２次関数を使ってグラフを描いてみる

$y = x^2$ という懐かしい響きのするグラフを書いてみましょう。まずは、横線で描くことを考えてみます。横線なので、２番目と４番目のパラメータが同じ値になります。

```
public void paint (Graphics g ) {
        int  i=1;
        while  ( i<=100 ) {
                g.drawLine( 1, i, i*i, i );   i++;
        }
}
```

終点のx座標をループ変数で指定しましたが、二乗するとあまりにも数が大きくなるので、すぐ表示領域からx

座標がはみ出してしまいます。しかも終点を動かしたので白黒の関係がなんとなく反対であるようみえます。

図6-11　終点を移動させた場合

そこで、終点のx座標を変更していましたが、逆に始点のx座標を変更の対象とし、かつ二乗した数を50ぐらいで割って、横方向の大きさを圧縮してみましょう。

```
public void paint (Graphics g ) {
        int  i=1;
        while  (  i<=100) {
                g.drawLine( i*i/50, i, 200, i );   i++ ;
        }
}
```

図6-12　始点を移動させ、かつ圧縮をした場合

頭を90度回転させれば、２次曲線に見えてきました。それでは、今度は縦線で濃く塗り潰すような形で書いていきましょう。まず、y方向がアプレットでは逆方向になっているので、最大値（ここでは仮に100としています）から引く形で描いてみましょう。つまり、$y = 100 - x^2$ という式になります。今度は、縦線なので始点と終点の x 座標が等しくなります。

```
public void paint (Graphics g ) {
        int  i=1;
        while  ( i<=100 ) {
                g.drawLine( i,  0,  i, 100  -  i*i/50 );   i++;
        }
}
```

図6-13　縦線で描いた場合

だんだん、昔教科書で見たような図形に近づいてきました。対称形にするために、x座標をマイナス-50から+50までの範囲を求めてみましょう。しかし、アプレット上のx軸は0から始ります。マイナスの部分は表示されません。そこで、中心を予め右にずらしましょう。ということで、x座標にいつでも+50を足します。

加えて、上の結果を見ると、x座標が100になる前に曲線の部分が終わってしまっています。これを避けるために、-50や+50の場合についてもう少し精密に考えてみましょう。xの値が50のときは、圧縮を掛けないとなると、y = 100 - 50 * 50 = 100 - 2500 = -2400となって、途方もなくアプレットの描画範囲からはみ出してしまいます。そこで、縦方向に圧縮を掛けましょう。-50や+50のときに２乗した結果がちょうど100になるようにすれば、y = 100 - 100 = 0 となって描画範囲からはみ出ません。そのための圧縮率は、100/2500すなわち、1/25ということになります。よって、最終的な２次関数は、$y = 100 - x^2/25$ということになります。これを終点のy座標として指定してみます。

```
public void paint( Graphics g ) {
        int x = -50;
        while ( x<= 50 ) {              //  xを-50から+50まで1つずつ変化させる
                g.drawLine( x+50, 0, x + 50, 100 - x * x / 25 );
                x++;
        }
}
```

上のメソッドでは、drawLineのx座標の値が始点・終点共に、x + 50という同一の値になっているので、縦線を引くことになり、かつその縦線は、y座標として、0から、２次関数のカーブを表す座標まで描かれます。これによって描かれる図は、次のような感じになります。

図6-14　最終的な２次曲線を持つ図形

drawLineメソッドを使って、繰返しを用いて図形を描くにはどのようにすれば良いかを考えてきました。このような簡単なメソッドでも、与えるパラメータの式を工夫することによって、いろいろな図形を描くことができるのです。アプレットの描画領域に収めるためには、圧縮をしたり、一定の数を足したりして、パラメータの式をうまく設計しないといけません。圧縮率や最大値などは、どの範囲を描きたいかによって自分で工夫してみてください。ちなみに、縦線ではなくて、始点をまったく１つの点にしてしまえば（すなわち、g.drawLine(50, 0, x + 50, 100 - x * x / 25);にすれば）、こんな渋い図形も描けます。

図6-15　始点を一点にした場合

６－３．様々な図を描くアプレットの例

drawLineと繰返しを使う例を学んできました。いろいろ描画メソッドと繰返しを組み合わせるとどんな図形が描けるのか考えてみましょう。各メソッドに与えるパラメータの式に注目してください。

★例題１　　対数グラフの方眼紙を描くアプレット

対数グラフとは、$y = e^x$ みたいなべき乗を基本にした関係があるときに、その関係を記すために利用する蔵うフのことです。ここでは、基数を２としています。変数 x の値の変化に注目してください。縦方向と、横方向の両方を対数グラフを描けるようにしています。例えば、縦方向は通常のグラフのための方眼紙にしたい場合はどうするのか考えてみなさい。

```
import  java.awt.* ;
import  java.applet.* ;

public  class  Exponential  extends  Applet  {
        public  void  paint( Graphics  g  ) {
                int  x= 1;
                white (  x <= 128 ) {
                        g.drawLine( x ,  1,  x,  128 );  g.drawLine( 1,  x,  128,  x  );
                        x = x * 2;
                }
        }
}
```

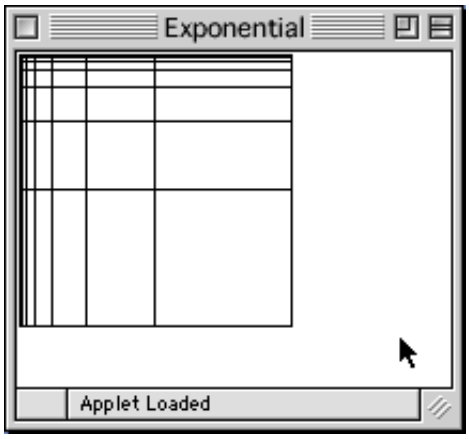


図6-16 対数アプレットの描画例

★例題２　四角形を描くアプレット

枠の中に枠を描いていきます。始点の座標と幅と高さが、変数 x を元に、どのように変化しているのか考えてみてください。ピラミッド上にしているので、幅と高さを、始点の座標の２倍の大きさで小さくしていっています。これを、２倍にしないで実行してみて、どう描画されるか確かめて下さい。余裕があれば、drawRectを、drawRoundRectに変えてみてください。その場合は、丸みの指定も忘れずに。

```
import  java.awt.* ;
import  java.applet.* ;

public  class  Pyramid  extends  Applet  {
        public  void  paint( Graphics  g  ) {
                int  x = 10;
                while ( x <= 60  )  {
                        g.drawRect(  x,  x,  180 -x*2,  180 -x*2 );
                        x = x + 10;
                }
        }
}
```

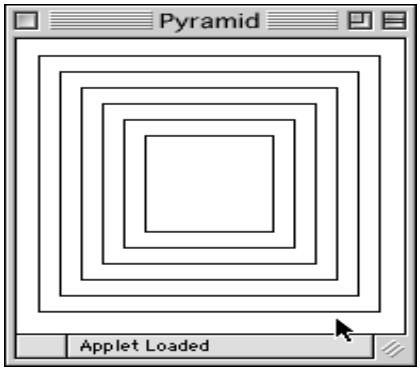


図6-17 ピラミッドアプレットの描画例

★例題３　円や弧を描くサンプル

何故、drawArcでは、始点の座標を変数 x が持つ値の半分にしているのでしょうか？例題２の始点の位置の変化と、幅と高さの変化の関係を元に考えてみなさい。また、開始角度と角度差も、半分の関係にあります。いろいろ、パラメータの式を変えてみて、どのように描画されるのか考察してみてください。

```
import  java.awt.* ;
import  java.applet.* ;

public  class  ArcCircle  extends  Applet  {
        public  void   paint( Graphics  g  ) {
                int  x= 10;
                while (  x  <=100 ) {
                        g.drawOval( 10, 10, x, x );
                        g.drawArc(  150-x/2,  150-x/2,  100+x, 100+x, -x/4,  x/2 );
                        x = x + 30;
                }
        }
}
```

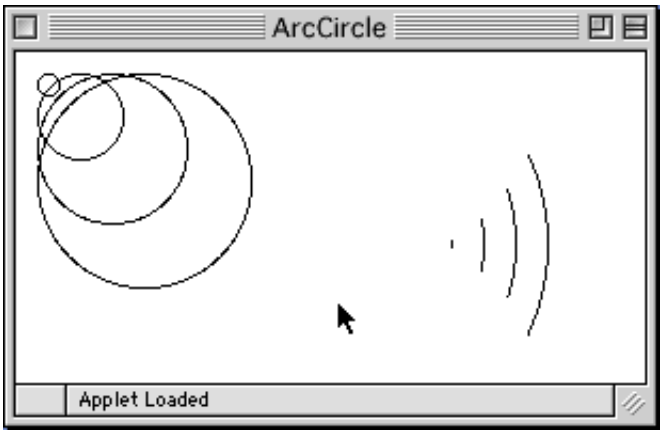


図6-11 円弧アプレットの描画例

６－４．while文の多重化（ネスト）

while文で「繰り返したいこと」の中に、さらにwhile文を書くことができます。このような「繰返しの中の繰返し」のことを、繰返しの多重化（Nesting）と呼んでいます。たとえば、次の例は簡単な繰返しの多重化の例です。

```
int     outer = 1;
while ( outer <= 4 ) {
        int     inner = 1;
        while ( inner <= 5 ) {
                System.out.print( "Thank you! " );
                inner += 1;                         // 内側のループ変数の更新
        }
        outer += 1;                                 // 外側のループ変数の更新
        System.out.println( "" );                   // 改行させている
}
```

変数outerが外側の繰返しのループ変数になっています。変数innerが内側の繰返しのループ変数になっています。外側の繰返しは、4回行なわれます。内側の繰返しは、5回繰り返されます。ですから、端末上にThank you!の文字列は、4 × 5回で、計20回表示されることになります。外側のループ変数の更新は、内側の繰返しのブロックが終わった後に、記述されていることに注意してください。この例は次のような表示をします。

```
Thank you!  Thank you!  Thank you!  Thank you!  Thank you!
Thank you!  Thank you!  Thank you!  Thank you!  Thank you!
Thank you!  Thank you!  Thank you!  Thank you!  Thank you!
Thank you!  Thank you!  Thank you!  Thank you!  Thank you!
```

次のアプレットプログラムの例は、同じように多重化された繰返しによって記述されています。この例について、もう少し詳しくみていきましょう。

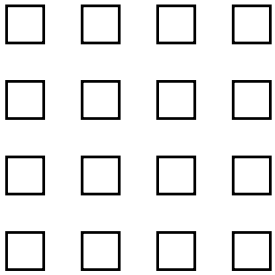
```
import  java.awt.* ;
import  java.applet.* ;

public class  LatticeDrawer extends  Applet  {
    public  void  paint( Graphics  g  ) {
        int     x, y;                           //  変数の宣言
        y = 10;
        while  (  y  <= 70  ) {                 //  縦方向の繰り返し
            x  = 10;                            //  内側のループ変数の初期化
            while  (  x  <=  70  )  {           //  横方向の繰り返し
                g.drawRect(  x,  y,  10,  10  );//  四角形を書く
                x  =  x  +  20;                 //  内側のループ変数の更新
            }
            y  =  y  +  20;                     //  外側のループ変数の更新
        }
    }
}
```

この例の意味を考えていくのに、まず変数 x と y が繰返しによってどのように変化していくかを探っていきましょう。変数 y の方が外側のwhile文の中で値が変えられているので、変化がゆっくりになっています。変数 x は、内側のwhile文の中で変化させられているために、変化は急速になっています。

いま、(x , y)という記述の仕方で、それぞれの変数が、内側のwhile文のdrawRectメソッドを呼び出す時点でどのような値を取っているか追いかけてみましょう。変数 x が頻繁に変化しているのがわかると思います。

(10, 10)→(30, 10)→(50, 10)→(70, 10)→
(10, 30)→(30, 30)→(50, 30)→(70, 30)→
(10, 50)→(30, 50)→(50, 50)→(70, 50)→
(10, 70)→(30, 70)→(50, 70)→(70, 70)

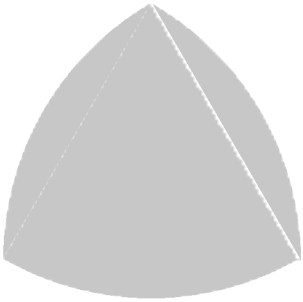
実際にどう書かれるのでしょうか？変数の値が、それぞれ四角形の開始位置（左上角）の座標になっていますので、こんな感じで描画されます。



６－５．課題

課題6-1.

弧を塗りつぶして描くメソッドを用いて、次のような図を描くようなアプレットを作成しなさい。クラス名は、Tripletとしなさい。

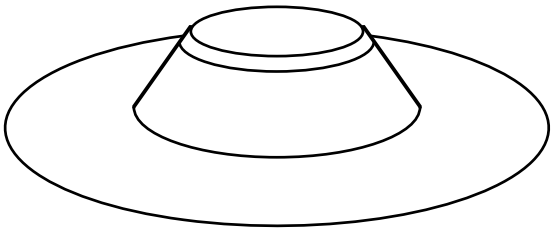


問題の図

課題6-2.

弧を描くメソッドと、直線を描くメソッド、楕円を描くメソッドを用いて、次のような図を描くようなアプレットを作成しなさい。クラス名は、Pudding（プリン）としなさい。

ヒント：弧の開始角度や、角度差を微妙に調整してみてください。

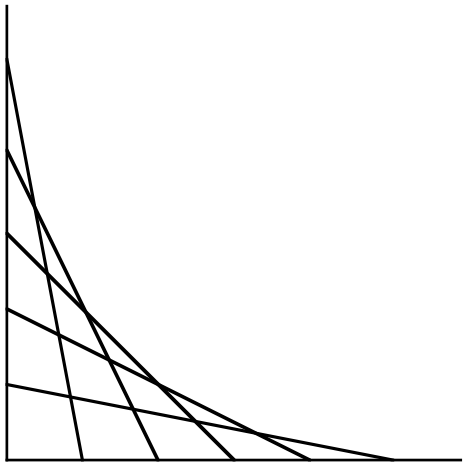


問題の図

課題6-3.

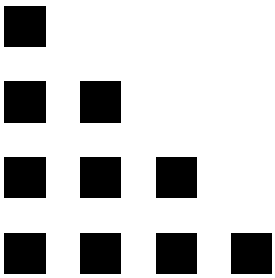
繰返しと、変数を１つまたは２つ使って、次のような図を書くアプレットを作成しなさい。高さ・幅は表示する位置は各自で決めて構いません。好みに応じて、色を変えても構いません。あるいは、線の数をもっと緻密に増やしても構いません。クラス名は、Russianとしなさい（1970年の万国博覧会のソ連館がこれに似ていたことから）。ただし、クラス名は別の名前にしても構いません。

ヒント：例えば終点のx座標は20ずつ増えて行くようにし、始点のy座標は20ずつ増えて行くようにすればいいのではないでしょうか？



問題の図

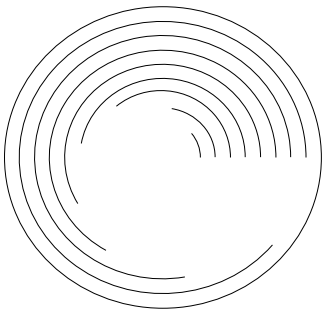
課題6-4.
問題の格子図を書くようなアプレットのプログラムを作成しなさい。while文の多重化の書き方を使って記述しなさい。クラス名はStairsにしなさい。



問題の格子図

ヒント：drawRectではなくfillRectを用いなさい。内側のwhile文の継続条件を工夫しなさい。

課題6-5.
繰返しを用いて問題の孤形図を書くアプレットのプログラムを作成しなさい。クラス名はArcsにしなさい。



問題の弧形図

ヒント：まずは、drawOvalを使って同心円を書くことから始めてみなさい。それができたら、drawArcを使って、弧を描きましょう。９つの弧があり、一番外側は、円になっていることから、角度差を割り出してみてください。