第８章．再帰処理

【学習のねらい】

① 再帰処理の概念を理解する。
② 実際の問題に再帰処理を応用する。

再帰処理とは、聞きなれない言葉だと思いますが、プログラミング関係では良く出て来る概念です。一言で言うと、自分の定義に自分自身（の定義）を含んでいる、ということになります。これだけでは、何のことか良く分からないと思いますので、ともかく具体的な例から学習して行きましょう。本章を学習すれば、再帰処理の意味、そしてそれがどういう場合に有効であるかが理解できるはずです。

８－１　再帰処理

最初の例として、ある数の階乗を求める処理を考えましょう。階乗とは数学上の演算ですが、その定義は簡単です。例えば５の階乗は５！と表し、

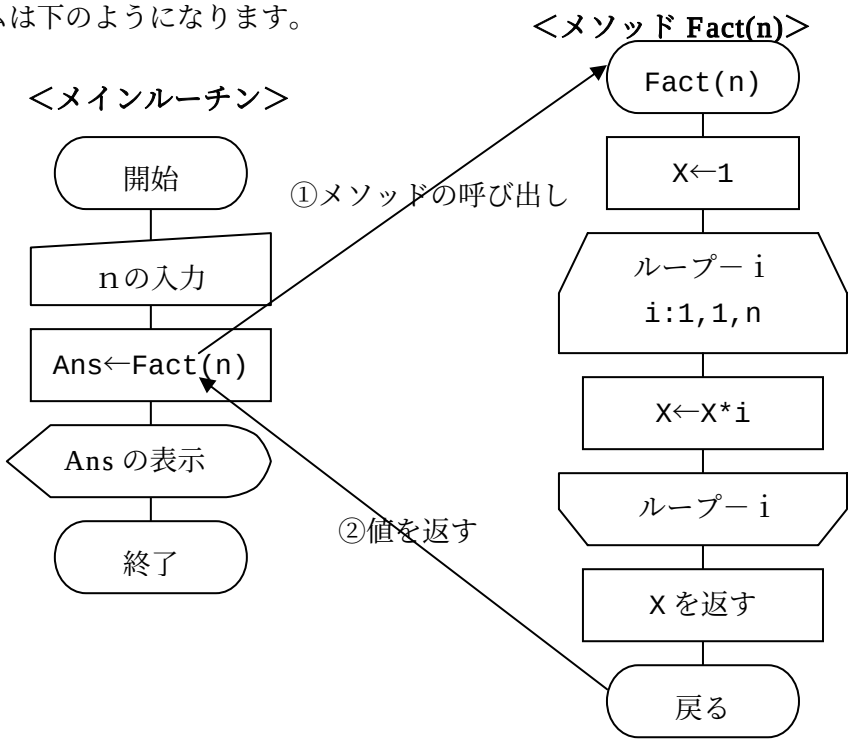
５！＝５×４×３×２×１＝１２０

となります。ですから、一般に正の整数ｎの階乗は

ｎ！＝ｎ×(n-1)×(n-2)×･･･×２×１

と表されます。

今、正の整数ｎを与えた時にそのｎ！を求めるプログラムを考えましょう。後の説明の都合上、n の階乗を求めるメソッド Fact(n)を定義して、これを用いることにします（階乗は英語で factorial といいます。そこで、その最初のスペルをとってメソッド名を Fact としました)。アルゴリズムは下のようになります。



前期テキスト第６章で学習した通り、（戻り値のある）メソッドを呼び出すと、その値を計算した後、再び（その値を伴って）呼び出したプログラムに戻ってきます。

簡単な処理ですから、問題なく理解できることと思います。ひとまず、この処理を行うプログラムを作成しましょう。
次の様なフレームを作成してください。

プログラムは次のようになります。メソッドの定義に関して不明な点があれば前期テキスト第６章（6-8 節）を参照してください。

＜［計算］ボタン＞

```
void jButtonKeisan_actionPerformed(ActionEvent e) {
  int Ans,n;
  n=Integer.parseInt(jTextFieldN.getText()) ;
  Ans=Fact(n); //関数 Fact(n)の呼び出し
  jTextFieldAns.setText(String.valueOf(Ans));
}
```

＜メソッド Fact(n)の定義＞

```
int Fact(int n) {
  int X;
  X=1;
  for (int i=1;i<=n;i++) {
    X=X*i;
  }
  return X;
}
```

作成したら、動作を確認してください。

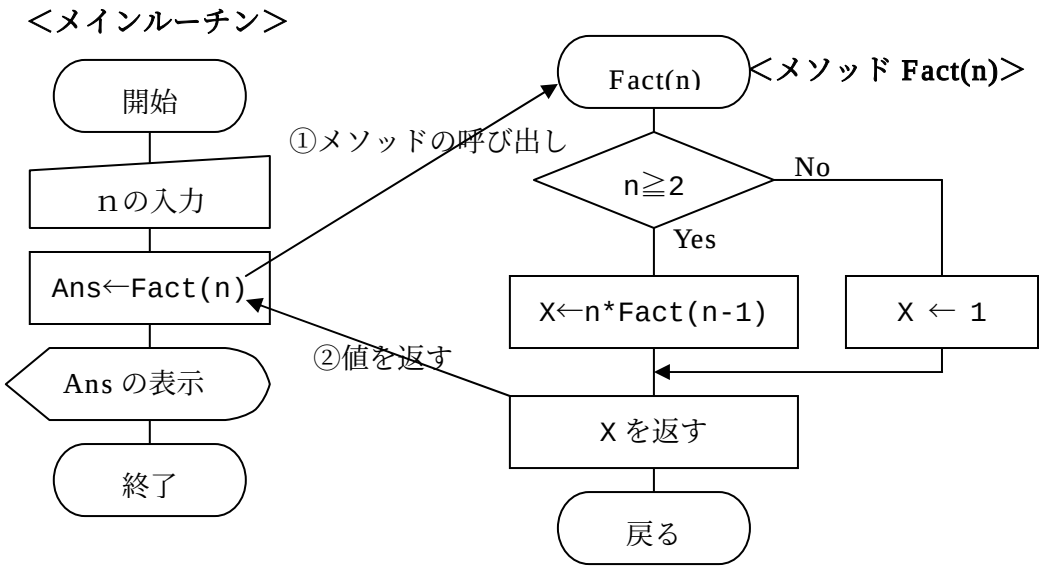
ここまでは、再帰処理とは何の関係もありませんでした。それでは、上のプログラムを再帰処理、より正確に言うと、メソッドの再帰的定義を用いて書き換えてみましょう。そのために、メソッド Fact(n)の定義を次のように捉え直します。階乗の定義から、メソッド Fact(n)は、

Fact(n) ＝n×Fact(n-1)

と表されます。つまり、メソッド Fact は（1 段階前の）自身の値を用いて定義する事ができるのです。これを再帰的定義と言います。この定義を用いると、Fact(n-1)はさらに Fact(n-2)を用いて表され、その Fact(n-2)はさらに Fact(n-3)を用いて表され・・・、という様に処理が順送りされ、最後に Fact(1)でこの連鎖は止まります。

この処理を流れ図で表すと次のようになります。p.123 の流れ図と比べてみてください。

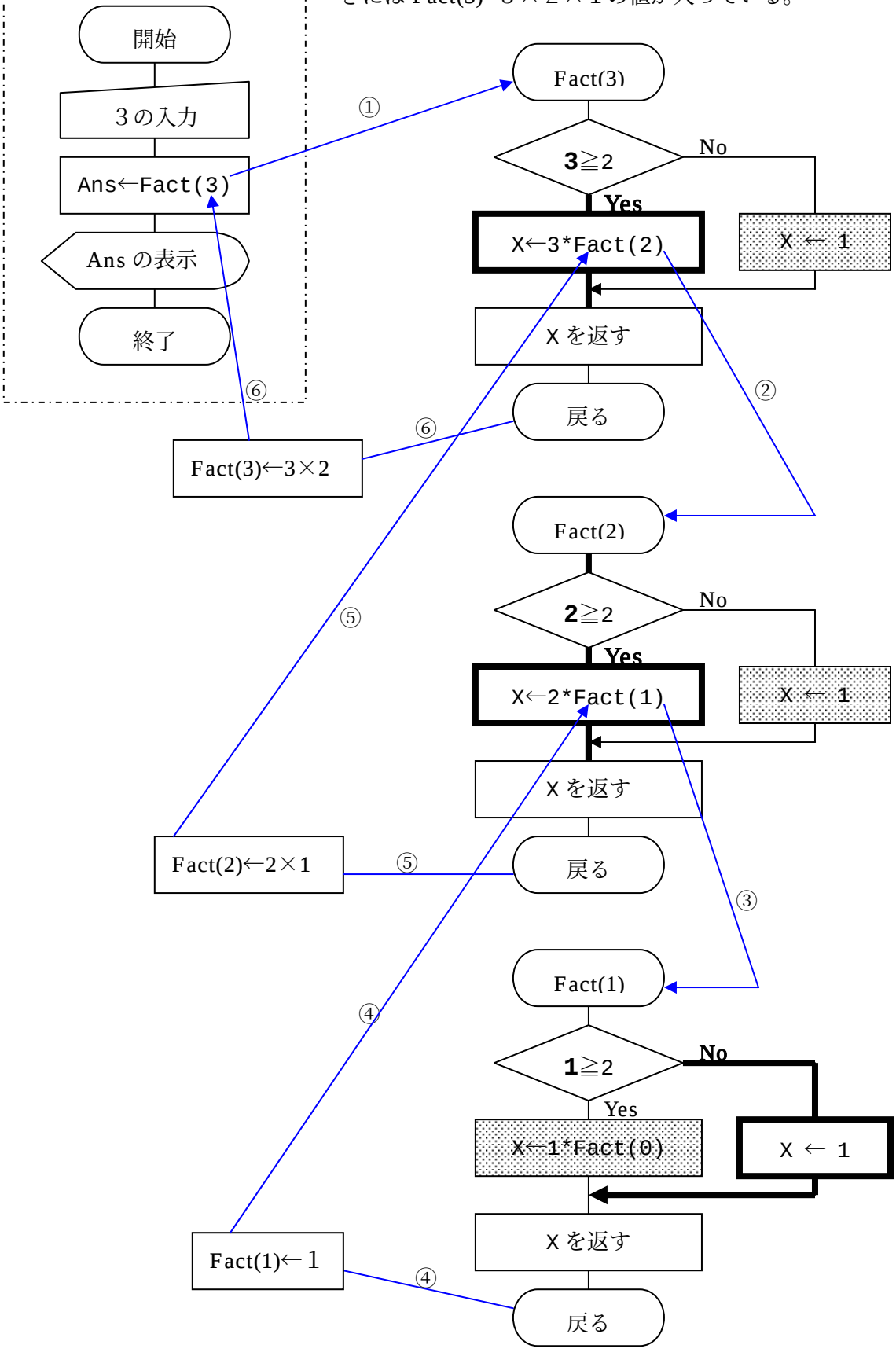
＜n！の計算＞－再帰的定義を利用



説明だけでは、なんとなく腑に落ちない部分が残ると思います。そこで、次ページに、3! を求める場合の処理の流れを具体的に展開しておきましたので、理解できるまでよく確認してください。

Fact(3)が呼び出され、それが①~③の流れを通じてFact(1)にまで到達する。
それから④~⑥の流れにより、Fact(3)に戻る。戻ったときには Fact(3)= 3 × 2 × 1 の値が入っている。

3!を求める処理の流れ



【基礎課題 8-1】

上で作成した、階乗を求めるプログラムの Fact(n)の部分を、p.125 の流れ図で示した様に再帰的定義を用いて書き直しましょう。空欄を埋めてください。

＜メソッド **Fact(n)**の定義＞

```
int Fact(int n) {
  int X;
  if (n>=2) {
    X= [                    ] ;
  }
  else {
    X=1;
  }
  return X;
}
```

作成したら実行し動作を確認して下さい。

【基礎課題 8-2】

メソッドの再帰的定義を適用するもう一つの例として、コラッツ（collatz）の予想という問題を採り上げましょう。これは数学上の問題で、未だに証明が完結していない大問題です。・・・と言うと難しそうですが、問題それ自体は簡単です。それは、

- ある数字が偶数なら２で割る
- 奇数なら 3 倍して１を加える

という操作を繰り返すと、必ず１になる、というものです。一つ試してみましょう。例えば５の場合

５→１６→８→４→２→１

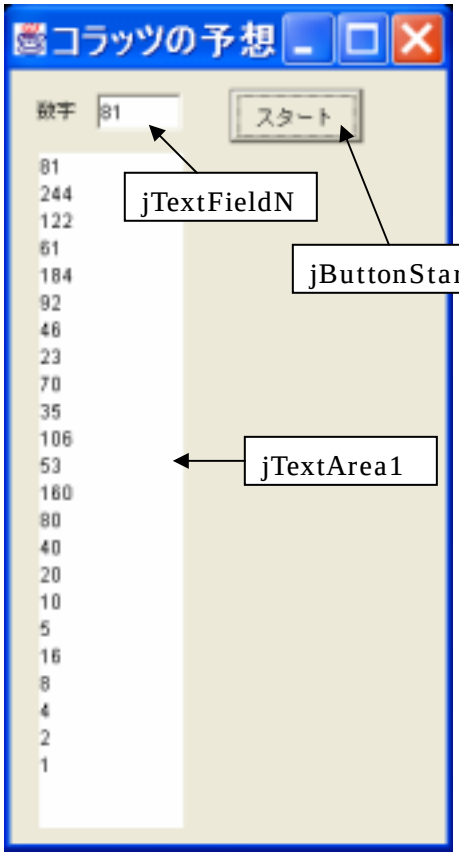
と確かに１になりました。今度は１１の場合は、

１１→３４→１７→５２→２６→１３→４０→２０→１０→５→１６→８→４→２→１

と少し手間がかかりましたが、確かに１になります。これを一般の整数について証明するのが上の問題ですが、それはまだ実現されていません。

しかし、我々にはコンピュータがあります。任意の整数ｎを入力したときにそれが 1 になるかどうかをコンピュータで確かめるプログラムを作りましょう。

作成するプログラムの動作内容は次の通りです。



プログラムを起動すると、左の様な起動画面が現れます。ここで、調べたい数字を入力し［スタート］ボタンをクリックすると、コラッツの操作を繰り返し、１になったら停止する、というものです。

この処理を行うために、ある整数ｎに対して

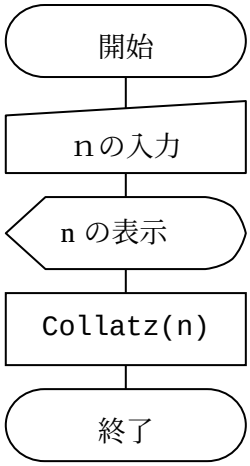
- ｎが偶数なら２で割る
- ｎが奇数なら 3 倍して１を加える

という操作を行うメソッド Collatz(n)を定義しましょう。

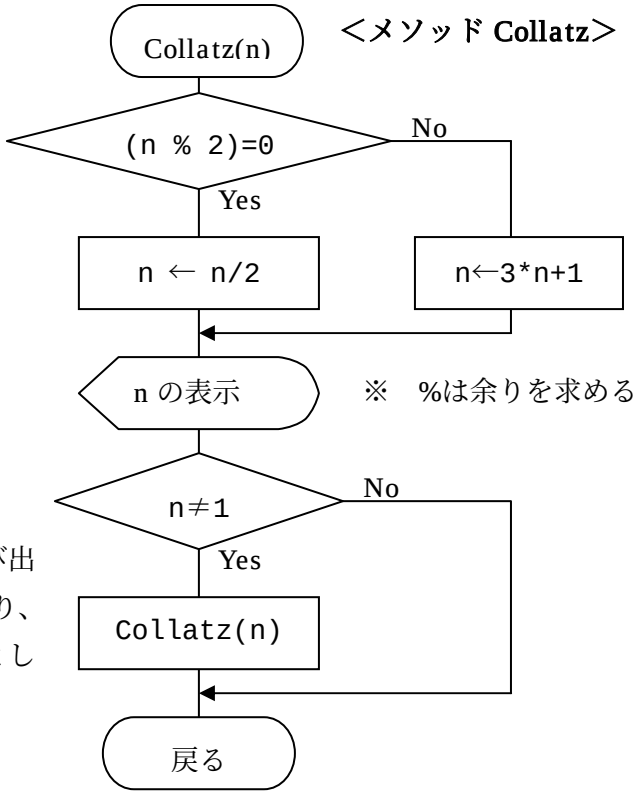
このときのプログラムの流れ図は次のようになります。

＜Collatz の操作＞

＜メインルーチン＞



＜メソッド Collatz＞



※　%は余りを求める演算です。

メソッド Collatz を再帰的に呼び出しています。こうすることにより、プログラムの記述がすっきりとします。

プログラムは次の通りとなります。作成したら実行し、動作を確認してください。

＜［スタート］ボタン＞

```
void jButtonStart_actionPerformed(ActionEvent e) {
  int n=Integer.parseInt(jTextFieldN.getText());
  String Disp=n+"¥n";
  jTextArea1.setText(Disp);  //jTextArea に最初の数字を表示
  Collatz(n);  //コラッツの操作を行うメソッドを呼び出す
}
```

＜メソッド Collatz の定義＞

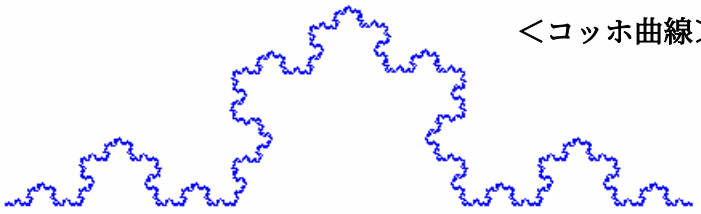
空欄を埋めてください。分からなければ前ページの流れ図を参照して下さい。

```
void Collatz(int n) {
  if( (n % 2)==0 ) {
    n=n/2;
  }
  else {
    n=3*n+1;
  }
  String Disp=jTextArea1.getText();
  jTextArea1.setText(Disp+n+"¥n");
  if (n!=1) {
    
  }
}
```

８－２　再帰処理の応用－フラクタル図形

前節の例で分かったと思いますが、再帰処理が有効なのは、同種の操作を繰り返し適用することで実現できる処理の場合です。実は、コンピュータグラフィックスの世界に、その格好の応用例があります。それは、フラクタル（自己相似）図形というものです。ここでは、その詳細は気にせず、ただ単純に「その図形のどの一部をとっても全体と同じパターン（形）になっているような図形」と捉えておくことにしましょう。もっとも、これだけでは、ピンと来ないかもしれません。そこで、さっそく具体例に進むことにします。

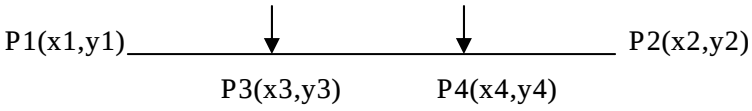
採り上げるのは、コッホ（Koch）曲線という図形です。下がその図形なのですが、何やら込み入った形をしていますね。



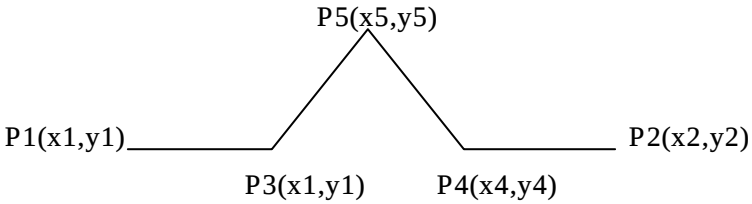
ところが、このコッホ曲線はある単純なパターンの繰り返しによって描かれたものなのです。種明かしをしましょう。この図形の描き方は次の通りです。

① まず、2 点 P1(x1,y1)、P2(x2,y2)を結ぶ直線からスタートします。

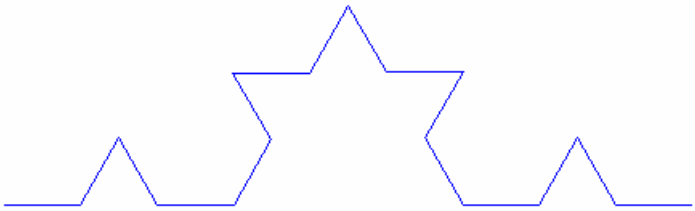
② 次に、この直線を 3 等分する点 P3(x3,y3)および P4(x4,y4)を求めます。



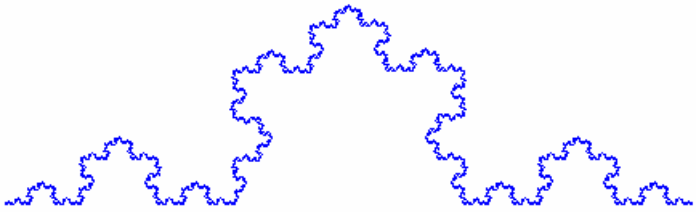
③ 線分 P3-P4 を底辺とする正三角形の頂点 P5(x5,y5)を求めます。さらに、直線 P1-P2 を下の様に４つの線分｛P1-P3，P3-P5，P5-P4，P4-P2｝に置き換えます。



④　続いて、それら４つの線分に対して同じ操作を行います。すると下のような図形になります。



同様の操作を各線分について繰り返して行くと、先ほど示したコッホ曲線が現れてきます。下は、上の操作を 7 回繰り返した時点での図形です。



このように、コッホ曲線は、「一つの線分を４つの線分に置き換える」という操作を、図形中の各線分に適用することで描かれます。つまり、コッホ曲線は、この「一つの線分を４つの線分に置き換える」という処理を再帰的に適用することで描かれる図形なのです。ここまで分かればコッホ曲線を描画するプログラムの作成は難しくありません。ただ、その前に、上の処理で必要になる、点 P3～P5 の座標（の表式）を求めておきましょう。

まず、点 P3、点 P4 の座標は次のように与えられます。これはすぐに分かると思います。

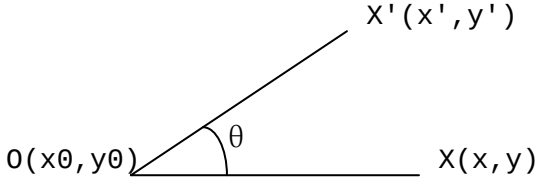
```
x3=x1+  (x2-x1)/3   =(2×x1+x2)/3
y3=y1+  (y2-y1)/3   =(2×y1+y2)/3
x4=x1+ 2×(x2-x1)/3 =(x1+2×x2)/3
y4=y1+ 2×(y2-y1)/3 =(y1+2×y2)/3
```

次に、P5 の座標は少しだけ複雑です。前ページにおける処理③の図より、点 P5(x5,y5)は点 P3(x3,y3)を中心として、P4(x4,y4)を 60 度回転して得られた点と考えることができます（正三角形の内角は 60°です）。このことより、P5 の座標は次のように求められます。*)

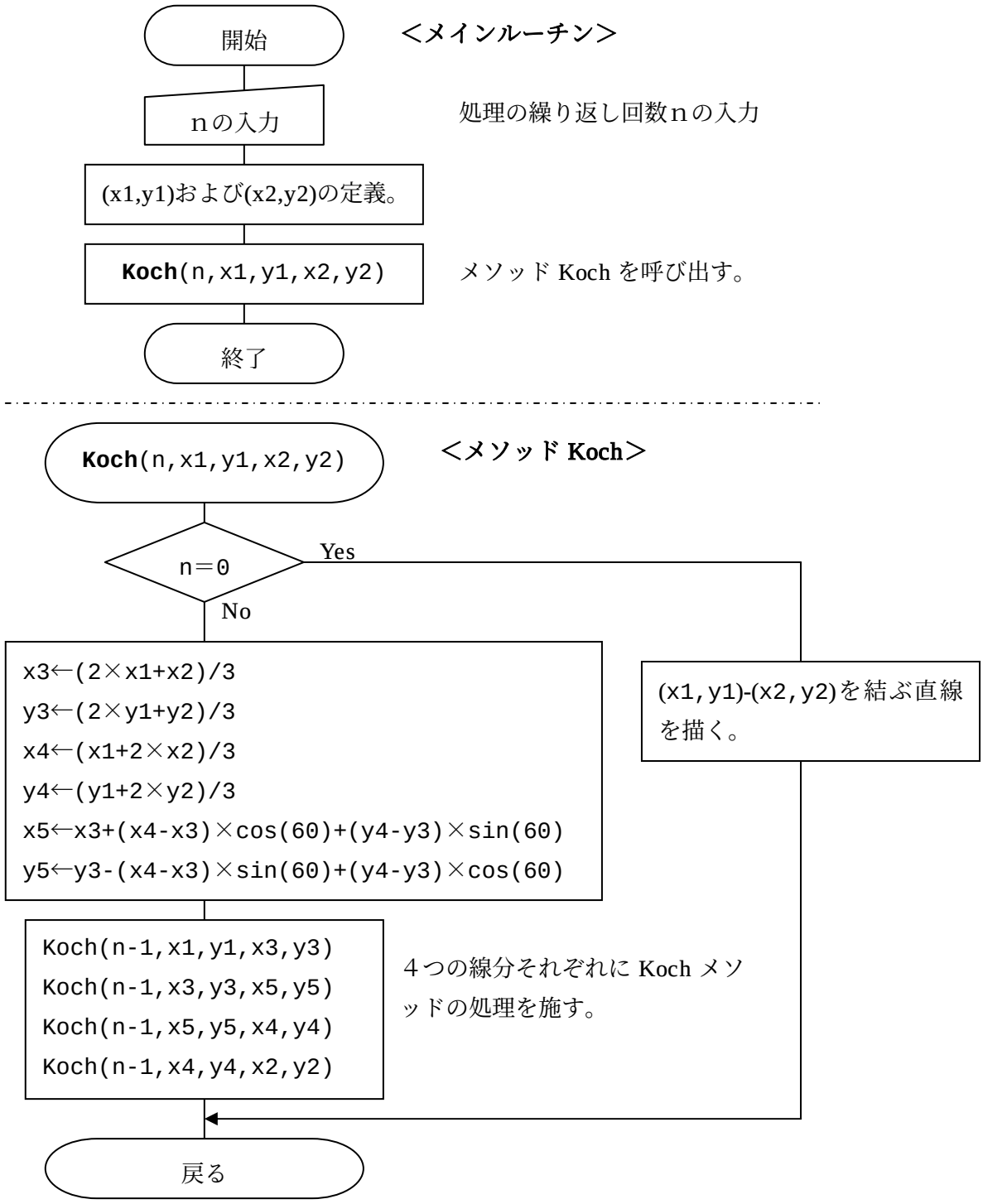
```
x5=x3+(x4-x3)×cos(60°)+(y4-y3)×sin(60°)
y5=y3-(x4-x3)×sin(60°)+(y4-y3)×cos(60°)
```

＊　）　一般に、点 O(x0,y0)を中心として点 X(x,y)を角度θだけ回転させて得られる点 X'(x',y')の座標は、次のように与えられます。

```
x'=x0+(x-x0)×cosθ+(y-y0)×sinθ
y'=y0-(x-x0)×sinθ+(y-y0)×cosθ
```



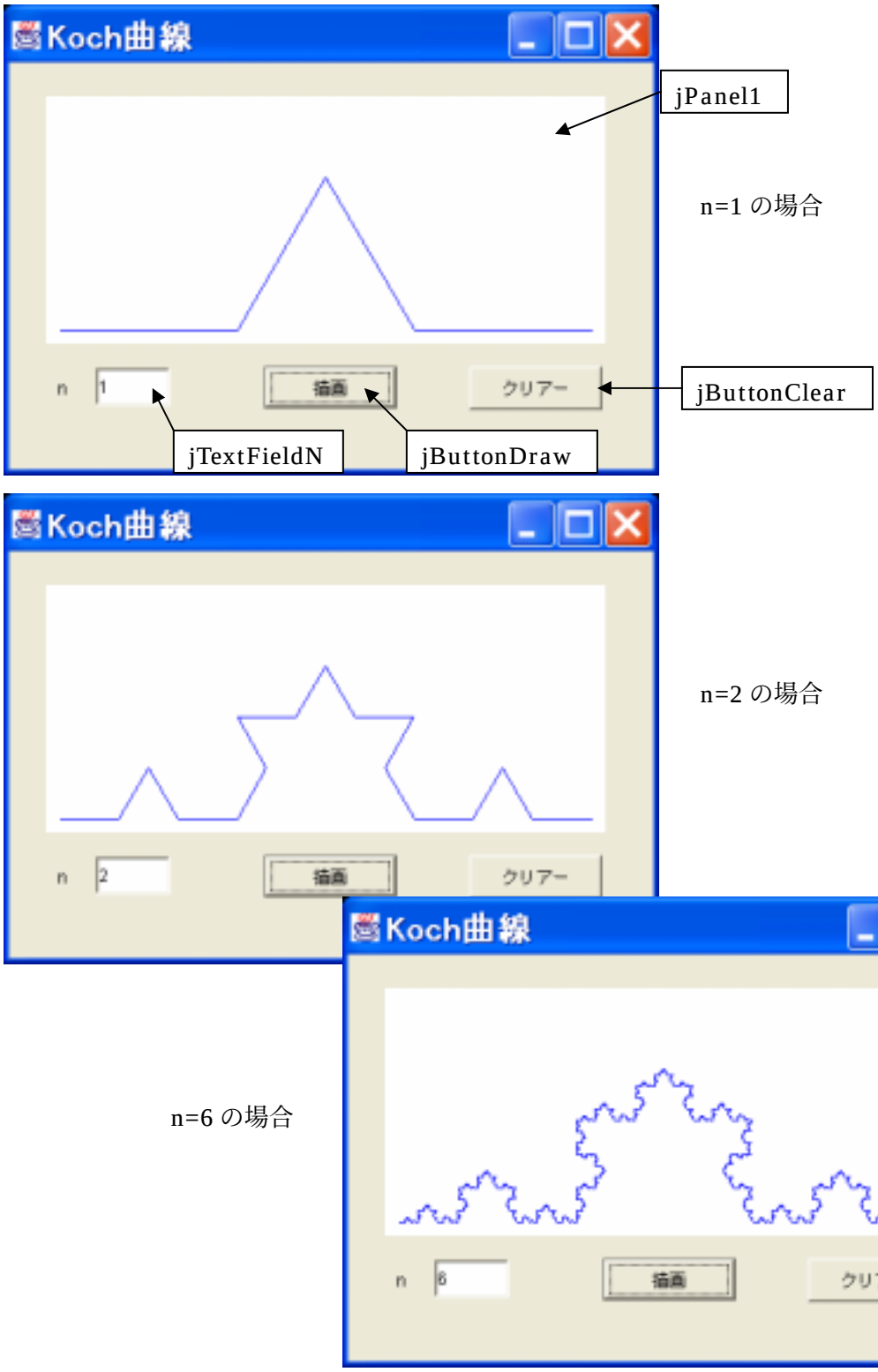
メソッドの再帰的定義を利用した、コッホ曲線・描画処理の流れ図をまとめておきます。



※　n=1,2,3 の場合について具体的に流れを追って、きちんと Koch 曲線が描かれるか確認してください。それが確認できれば、この処理を理解できたことになります。

【応用課題 8-A】

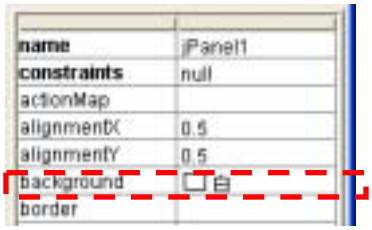
コッホ曲線を描くプログラムを作成しましょう。動作内容は次の通りです。
プログラムを実行し、現れた画面上で、下の様に、繰り返し回数ｎの値を入力し［描画］ボタンをクリックすると、図形が描かれます。



［クリアー］ボタンをクリックすると、パネル内の描画が消去されます。

HP の該当部に実行ファイル「Koch.exe」を掲載しています。適宜ダウンロードして動作内容の確認に利用してください。

それでは、プログラムの作成に取り掛かりましょう。まず、上のようなフレームを作成して下さい。そしてパネルコンポーネント（Swing Containers タグ内の左端にあります）については、background プロパティを白に変更して下さい。



［描画］ボタンのイベントハンドラは次の様になります。

＜［描画］ボタン＞

```
void jButtonDraw_actionPerformed(ActionEvent e) {
  int n=Integer.parseInt(jTextFieldN.getText()); //n の宣言・代入
  int XMax=jPanel1.getWidth();  //パネルの幅の取得          ①
  int YMax=jPanel1.getHeight(); //パネルの高さの取得
  int x1=10,x2=XMax-10;
  int y1=YMax-10, y2=YMax-10;                              ②
  Koch(n,x1,y1,x2,y2);  //Koch 曲線を描くメソッドの呼び出し ③
}
```

＜プログラムの解説＞

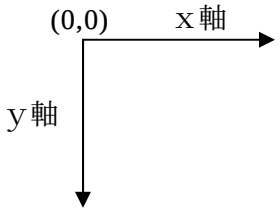
① パネルコンポーネントの幅と高さは getWidth()および getHeight()メソッドで取得できます。

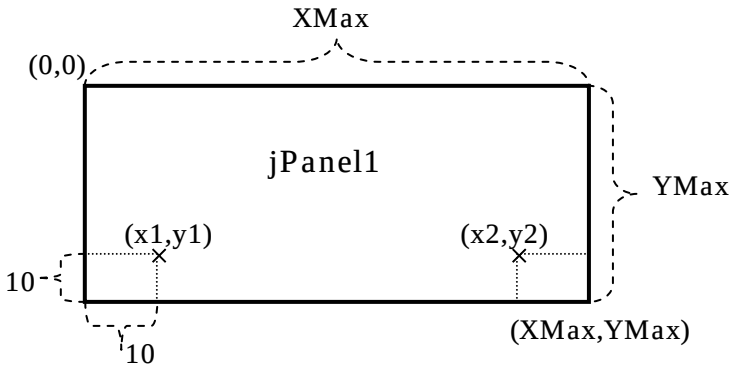
② p.130 で説明した最初の２点 P1(x1,y1)、P2(x2,y2)の座標の値を与えています。→次ページの【コンピュータでの座標の取り方】参照

③ Koch 曲線を描くメソッドを呼び出します。具体的な定義は p.136 で与えます。

【コンピュータでの座標の取り方】

コンピュータでは、右のように、左上を原点（0,0）として、横方向にｘ軸、そして縦方向にｙ軸をとります。数学の座標と異なりｙ軸が下向きになる点に注意して下さい。この約束に従えば、上のプログラムで指定した２点（x1,y1)、（x2,y2）の座標は下の通りとなります。





［クリアー］ボタンのイベントハンドラは次の様になります。

＜［クリアー］ボタン＞

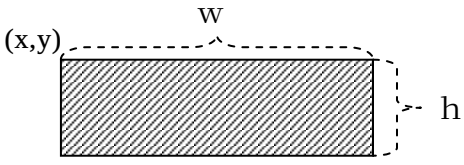
```
void jButtonClear_actionPerformed(ActionEvent e) {
  Graphics g=jPanel1.getGraphics(); //Graphics オブジェクトの取得  ①
  int Width=jPanel1.getWidth();    //パネルの幅の取得
  int Height=jPanel1.getHeight();  //パネルの高さの取得
  g.setColor(Color.white); //使用する色を白色に指定              ②
  g.fillRect(0,0,Width,Height); //パネルを白で塗りつぶす       ③
}
```

＜プログラムの解説＞

① Java 言語でグラフィックスを描画するには、Graphics クラスのオブジェクトを用います。そして Graphics オブジェクトは getGraphics()メソッドにより取得する事ができます。ここでは、パネルコンポーネントの Graphics オブジェクトを取得し、それに g という名前をつけています。描画に必要なメソッドは全て Graphics オブジェクトに用意されています。→詳細は、前期のプログラミング応用課題Ⅲ参照（http://ext-web.edu.sgu.ac.jp/HIKO/Prog/の応用編課題）

② `setColor()`メソッドにより、描画に使用する色を指定できます。色は Color クラスに定義されており「`Color.white`」は白色を意味します。

③ `fillRect(x,y,w,h)`は、下のように、（x,y）を左上隅として、幅w、高さｈの長方形領域を指定色（今の場合白）で塗りつぶすメソッドです。これにより、パネルは真っ白に塗りつぶされます。



最後に、コッホ曲線を描くメソッド Koch の定義は次のようになります。空欄を埋めてプログラムを完成させて下さい。分からない場合は p.132 の流れ図を参照して下さい。

＜Koch メソッド＞

```
void Koch(int n,int x1,int y1,int x2,int y2) {
  Graphics g=jPanel1.getGraphics(); //Graphics オブジェクトの取得
  g.setColor(Color.blue); //使用する色を青色に指定
  int x3,y3,x4,y4,x5,y5;
  if(n==0) {
    g.drawLine(x1,y1,x2,y2);  //(x1,x2)-(x2,y2)を結ぶ直線を描く  ①
  }
  else {
    x3=(2*x1+x2)/3;
    y3=(2*y1+y2)/3;
    x4=(x1+2*x2)/3;
④   y4=(y1+2*y2)/3;
    x5=x3+(int) ( (x4-x3)*Math.cos(60*Math.PI/180)
                  +(y4-y3)*Math.sin(60*Math.PI/180) );
    y5=y3+(int) (-(x4-x3)*Math.sin(60*Math.PI/180)
                 +(y4-y3)*Math.cos(60*Math.PI/180) );
    Koch(n-1,x1,y1, [  ] , [  ] );

    Koch(n-1,x3,y3, [  ] , [  ] );

    Koch(n-1,x5,y5, [  ] , [  ] );

    Koch(n-1,x4,y4, [  ] , [  ] );
  }
}
```

②　③

＜プログラムの解説＞

① drawLine(x1,y1,x2,y2)メソッドは、２点（x1,y1）－（x2,y2）を結ぶ直線を描くメソッドです。

② Java 言語では、sin や cos などの三角関数は Math クラスに定義されています（前期テキスト 6-1 節 p.148 参照）。例えば sin(x)は Math.sin(x)と表されます。

③ cos や sin の三角関数では、角度を度（°）ではなく、ラジアンで表します。180°がπラジアンに相当します。ですから 60°は（60×π/180）ラジアンに相当します。なお円周率πは Math クラスに定義されており、Math.PI と表されます。

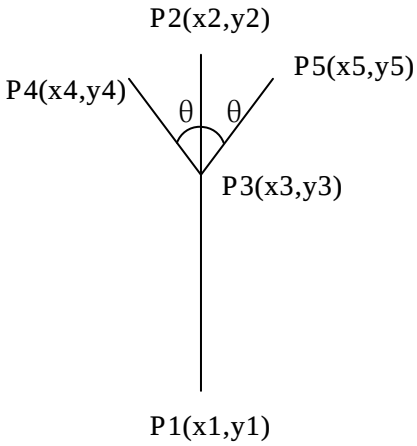
④ 三角関数の値は一般に実数になります。Java 言語では、実数の値を整数型の変数に代入しようとすると警告のエラーメッセージが出ます。そこで、(int)によって整数に**型キャスト**しています。→前期テキスト p.72 参照

作成したら実行し動作を確認して下さい・・・。Koch 曲線が描けましたか？

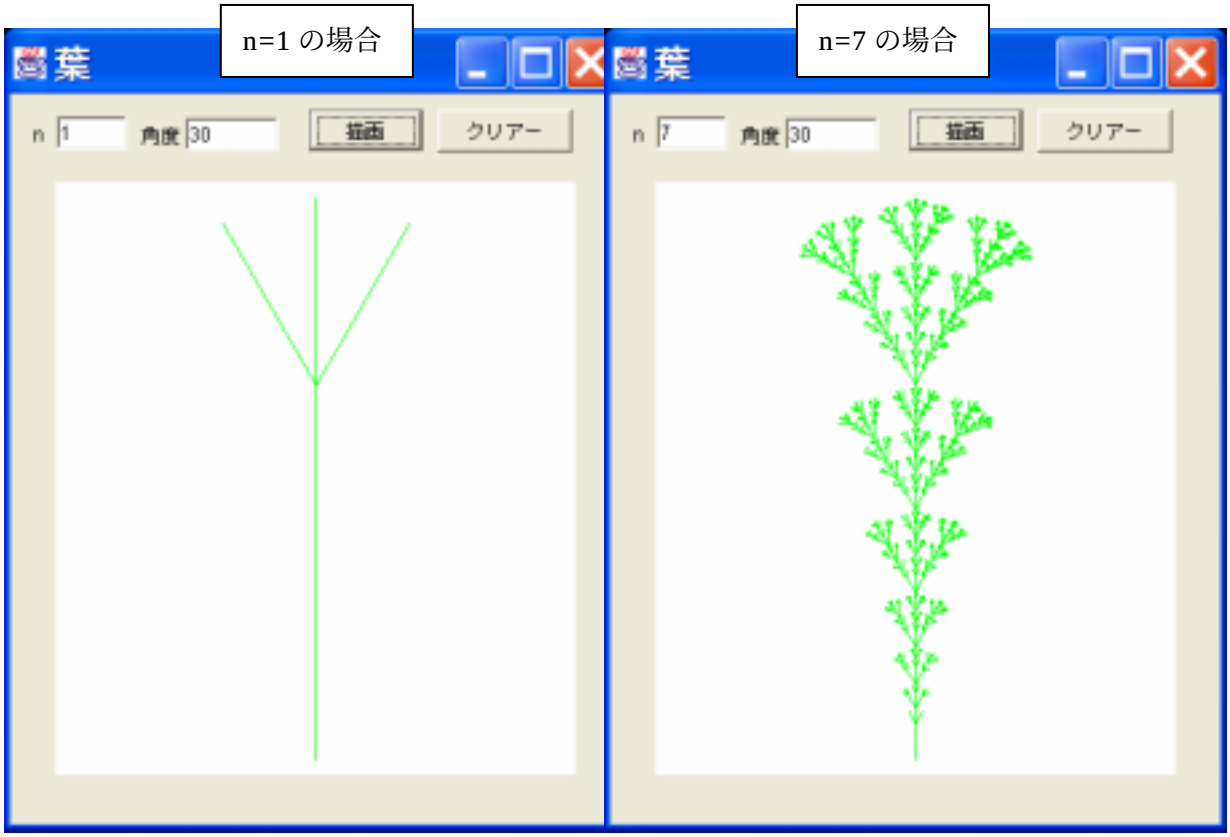
【応用課題 8-B】

今度は、植物らしきフラクタル図形を描いてみましょう。
この場合の操作は次の通りです。

① ２点 P1、P2 を結ぶ直線からスタートする。
② 線分 P1-P2 を 2:1 に内分する点 P3 を求める。
③ 続いて点 P3 を中心として、点 P2 を角度θだけ回転させて得られた点を P4 とする。同時に、－θだけ回転させて得られた点を P5 とする。
④ 線分 P1-P2 を４つの線分｛P1-P3，P3-P4，P3-P5，P3-P2｝に置き換える。
⑤ ４つの線分それぞれに、①～④の操作を行う。



以上の操作を繰り返すと、植物らしきフラクタル図形を描画することができます。このプログラムは【応用課題 8-A】を少し手直しすることで作成できます。作成するプログラムの動作例は次の通りです。プログラムを実行し、繰り返し回数と角度θの値を入力して［描画］ボタンをクリックすると下の様に図形を描きます。



やはり HP に実行ファイル「Leaf.exe」を掲載しています。必要な場合はダウンロードして動作を確認してください。

＜ヒント＞

（x3,y3）、(x4,y4)、(x5,y5)の表式は次のようになります。ただし、角度θは度（°）で与えられているものとします。

```
x3=(x1+2*x2)/3
y3=(y1+2*y2)/3
x4=x3+(x2-x3)*cos(θ*π/180)+(y2-y3)*sin(θ*π/180)
y4=y3-(x2-x3)*sin(θ*π/180)+(y2-y3)*cos(θ*π/180)
x5=x3+(x2-x3)*cos(-θ*π/180)+(y2-y3)*sin(-θ*π/180)
y5=y3-(x2-x3)*sin(-θ*π/180)+(y2-y3)*cos(-θ*π/180)
```