5.2 漸化式

では、フィボナッチ数列に戻ろう。

よく数列の n 番目の項を a_n などと表記することがある。すると次の $n+1$ 番目の項は a_{n+1}、$n+2$ 番目の項は a_{n+2} という書き方になるだろう。フィボナッチ数列は、直前の 2 項の和が次の項になっているので、一般に

$$a_{n+2} = a_{n+1} + a_n \tag{1}$$

という書き方ができる。とくにいまは 1, 1 から数列を始めているので、$a_1 = 1$, $a_2 = 1$ の条件が (1) に付け加わることになる。

これだけの条件と式があれば、この先の項が順次計算できるのだ。(1) のような形式で与えられる式を**漸化式**と呼ぶ。数学の世界に限らず、前後の関係は分かっているのだが、そのものズバリを与える式が不明であることは多い。実はフィボナッチ数列もそんなものの一つだ。数列の前後の関係は (1) により明確に分かっている。では、フィボナッチ数列のズバリ第 n 項を求める式は何だろう。これはちょっと複雑な計算をするので、詳しいことは省くが、フィボナッチ数列の第 n 項 a_n は

$$a_n = \frac{1}{\sqrt{5}} \left\{ \left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right\}$$

で与えられる。

実際に計算すると大変だが $n = 1, 2, 3, \ldots$ を代入すれば

$$1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ \ldots$$

が現れるのだ。式には $\sqrt{5}$ があるのに、フィボナッチ数列には一切登場しない点が興味深い。近似値でよければ、{ } 内の前の項が $(1.618033\ldots)^n$ で、後ろの項が $(0.618033\ldots)^n$ であることを利用しよう。n が大きいほど $(0.618033\ldots)^n$ は 0 とみなせるので、第 a_n 項は近似的に $1/\sqrt{5} \cdot (1.618033\ldots)^n$ と考えてよい。電卓を使えば、$n = 9$ のときは $a_n \approx 33.994116$ であることが分かる。たしかに第 9 項の 34 に一致している。

Java には漸化式をうまく扱える仕組みが備わっている。以前、フィボナッチ数列を生成するプログラム `[Fibonac2.java]` を覚えているね。そして君たちは `[Fibonac2.java]` をもとに、整数 n

を入力すると n 番目のフィボナッチ数を表示するプログラムを作っただろう。え？　作ってないだって？　だめだなあ、手抜きをしちゃあ。まあ、いいや。ここでは当時とまったく同じ動作をするものを、漸化式から作ってみよう。次のプログラムも、整数 n を入力すると n 番目のフィボナッチ数を表示する。

programming list [NthFib.java]

```
 1: import java.util.Scanner;
 2:
 3: public class NthFib {
 4:
 5:     public static void main(String[] args) {
 6:         Scanner s = new Scanner(System.in);
 7:
 8:         System.out.print("input a number: ");
 9:         int num = s.nextInt();
10:
11:         System.out.println("the " + num + "th Fibonacci number is " + Fib(num));
12:     }
13:
14:     public static int Fib(int n) {
15:         if(n > 2) {
16:             return (Fib(n - 1) + Fib(n - 2));
17:         } else {
18:             return 1;
19:         }
20:     }
21: }
```

これは少々親切な表示をするプログラムである。画面の入力要求 "input a number: " に対し、たとえば 10 を入力すると、"the 10th Fibonacci number is 55" と返してくれる。以前のプログラムに較べて複雑な印象を受けるのは、親切な表示をしたためではない。`Fib()` メソッドのせいである。

プログラムの仕組みを説明しておこう。まず先に、プログラムの本体である `main()` メソッドを見ていくことにする。これは実質 1 行のプログラムでしかない。

9:行目で、何番目かのフィボナッチ数を特定するための変数 `num` に、テキストフィールドからの値を取り込んだ。いままでは `int n` などと書いてきたが、少々味気ないので意味が分かりやすい変数名にしてみただけだ。

その結果、11:行目の `System.out.println;` 文で "the **th Fibonacci number is **" のように画面に表示されるのである。プログラムはこれ 1 行で済んでいるようなものだ。はじめの ** には、

入力した番号が出力されるのは分かるだろう。問題は後ろの ** である。ここには `Fib(num)` の値が表示されるはずなのだが、これは一体どのように計算されるのだろうか。

具体的に `num = 4` が与えられたとして処理を追ってみよう。処理は 14:行目からの `Fib()` メソッドの受け持ちだ。

9:行目で 4 の入力を受け取った **Java** は、11:行目の命令により、画面に "the 4th Fibonacci number is Fib(4)" と表示したくなっている。

一方 `Fib(4)` は、`Fib()` メソッドにより計算がされる。このとき `Fib()` が受け取る 4 は `n` の値として受け取るわけで、それは条件文 `if` の `n > 2` を満たしているから、素直に 16:行目の `Fib(n - 1) + Fib(n - 2)` に代入され、`Fib(3) + Fib(2)` が返される。あれれ？　いまは `Fib()` の計算のはずだ。なのに `Fib()` の計算に `Fib()` を使うのかい？　疑問はもっともだが、それでよいのである。もやもやを抱えつつも先へ行こう。

`return;` 文で `Fib(3) + Fib(2)` が返された結果 **Java** は、今度は "the 4th Fibonacci number is Fib(3)+Fib(2)" と表示する必要に迫られる。しかし、`Fib(3)` は再び `Fib()` メソッドの 16:行目の処理により `Fib(2) + Fib(1)` にして返されてしまうのだ。

ところが、`Fib(2)` の方は `n > 2` の条件を満たさないので、こちらは 17:行目の `else` により `1` が返される。その結果 **Java** は "the 4th Fibonacci number is Fib(2)+Fib(1)+1" を表示しようとする。

しかしながら、まだ `Fib(2)` と `Fib(1)` はメソッドを呼んで計算を続けるので、さらに `Fib()` により `1` と `1` が返ってくる。これでようやく **Java** は "the 4th Fibonacci number is 1+1+1" を表示すればよいことが分かり、結局 "the 4th Fibonacci number is 3" となるのだ。

一見すると詐欺にあったような印象を受けるだろうか。このような呼び出しは**再帰呼び出し**という。前後関係がはっきりしていて、一般の式を求める必要がないときなどに有効である。ただし、再帰呼び出しは計算量が爆発的に増加してしまう。たとえば `Fib(6)` の計算でも

```
Fib(6)  =  Fib(5) + Fib(4)
        =  {Fib(4)+Fib(3)} + {Fib(3)+Fib(2)}
        =  {(Fib(3)+Fib(2)) + (Fib(2)+Fib(1))} + {(Fib(2)+Fib(1)) + 1}
        =  ···
```

のように、`Fib(6)` が `Fib(5)`, `Fib(4)` を呼び、`Fib(5)` と `Fib(4)` がそれぞれ `Fib(4)`, `Fib(3)` と

`Fib(3)`, `Fib(2)` を呼び、さらにそれぞれが ...。何ということだ。呼び出しが倍々に増えているじゃないか。

TRY! あまり大きな数を与えるととんでもないことになるから、とりあえず `Fib(40)` 程度の計算をさせてみよ。さて、君たちのコンピュータは即座に結果を表示してくれただろうか。

おそらく即座に結果を返してくれまい。もし、一瞬のうちに結果が返る高速コンピュータを使っているなら、うらやましい限りだ。理由はこうだ。

`Fib(2)` や `Fib(1)` は二つの `Fib()` メソッドを呼ぶわけではないので、単純に `Fib()` 計算が 2^{40} 回行われることはない。実際はもっと少なく、2^{27} 回ほどだ。ひとくちに 2^{27} 回と言うものの、これでも軽く 1 億回を超えている。1 秒間に 1 億回の呼び出しが可能なコンピュータでも 1 秒はかかるので、"即座" に結果が出るわけではない。われわれが普段使っているコンピュータなら、少なくとも数秒は要するはずだ。再帰呼び出しをうかつに利用するのは要注意である。