1.2 10 進数と 256 進数

[Fibonacc.java] で試して分かったと思うが、フィボナッチ数列は結構大きな数になってしまう。君たちのコンピュータでは、何項目まで求めることができただろうか。コンピュータのことだから、無限に大きな値を扱うわけにはいかないが、やりようによっては数万桁の数でも扱える。しかし、この話題はしばらくあとになる。

フィボナッチ数の計算に使った変数は int 型の整数だったはずだ。コンピュータが扱える数値に限界があるのは、変数の型による制約が大きい。われわれが日頃使っている数は 10 進数である。これは 1 桁あたり 10 種類の数を使える。携帯電話は 090-xxxx-xxxx の番号を持っているので、自由になる桁は 8 桁ある。各桁に 10 種類の数が使えるから、電話番号は

$$10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 = 10^8 = 1 \text{ 億}$$

まで OK だ。残念ながら、国民一人ひとりには行き渡らない計算だが。

さて、 **Java** の "桁" にあたる単位は "バイト" という。1 バイトは xxxxxxxx の 2 進数で処理されている。具体的には 11010010 や 01011011 などという種類の "数字" が使えるのだ。すると 1 バイトで $2^8 = 256$ 種類の "数字" になる。したがって、 **Java** は 256 進数ということなのだ。何だか目が回りそうになってないかい？　気をたしかに持とう。

そこでようやく int 型の整数の話に入れる。int 型の整数は大抵 4 バイトである。4 バイトなら $256^4 = 4294967296$ 通りの数が使える。しかし int 型は整数を前提としている—つまり半分が正の数に、残りの半分が負の数に使われる—ので、4 バイトならざっと −2147483648〜2147483648 の整数が扱えるのだ。

大きな数を扱いたければ桁数を増やせばよいのだから、バイト値を増やせばいくらでも大きな桁を扱えることになる。もっとも、無制限に大きな桁を扱えるほど、型が用意されているわけではないのだけれど。int 型より大きい桁を扱いたいときは long 型を使えばよい。これは大抵 8 バイトの大きさだ。逆に int 型ほど大きな数を扱わないと分かっているなら、short 型を使うとよい。short 型は 2 バイトであるから、$256^2 = 65536$ 通りの数が使える。もちろん、これも正負の数で分け合うのだけれど。

さて、より大きなフィボナッチ数を見たければ、[Fibonacc.java] のプログラムで int 型の変数

を long 型で初期化すればよい。ただ、それだけでは [Fibonacc.java] 同様、一回実行するだけで終了してしまう。SUP の値をいろいろ変えて試したいときは、いちいちプログラムを変更しなくてはならず面倒だ。そこで、プログラムを実行してから SUP を決めるようにし、ついでに繰り返し SUP の変更ができるようにしてみた。

programming list [Fibonac2.java]

```
 1: import java.util.Scanner;
 2:
 3: public class Fibonac2 {
 4:
 5:     public static void main(String[] args) {
 6:         Scanner s = new Scanner(System.in);
 7:
 8:         for(;;) {
 9:             System.out.print("input SUP: ");
10:             final long SUP = s.nextLong();
11:             long f1 = 1, f0 = 1;
12:
13:             for(int i = 3; i <= SUP; i++) {
14:                 System.out.println(f1 + f0);
15:                 f1 = f1 + f0;
16:                 f0 = f1 - f0;
17:             }
18:         }
19:     }
20: }
```

[Fibonacc.java] との違いは 10:行目で long 型に変更したことで、メソッドも nextLong() を用いている。目を引くのは 8:行目の for 構文だろう。これは 18:行目までの繰り返しになっているが、for(;;) の中に条件が書かれていない。実は、このように条件を書かないことで繰り返しが常に行われるのである。**C** 言語などではお馴染みである。これにより、上限 SUP を入力し結果を表示することが延々繰り返される（途中でやめたくなったら、大抵 ctrl+c や ctrl+z でプログラムを中断させられる）。

TRY! "input SUP: " に対して上限を大きくし、より先のフィボナッチ数を求めてみよ。**Java** は、どれくらい大きなフィボナッチ数まで計算できるだろうか？

TRY! 整数 n を入力すると n 番目のフィボナッチ数だけを表示するよう、プログラムを変更せよ。

一番目の **TRY!**では、何かおかしなことが起きなかっただろうか。それはわれわれにはおかしく見えるが、**Java** にとっては自然なことなのである。

説明のため、コンピュータが内部で処理する数について触れておこう。いま、256 進数などと大袈裟なことを書いたが、コンピュータは内部では 2 進数で計算している。2 進数は 0 と 1 の 2 種類の数字で様々な数を表現する。だから 2 進数には、101 とか 11100 とか 10110111 とかの表現しかできない。これらは 10 進数では、それぞれ 5, 28, 183 を表している。むむ。では、正の数と負の数の区別はどこでするんだろう？

さっき、1 バイトが xxxxxxxx の 2 進数で処理されると言ったのを覚えているだろうか。この場合、コンピュータは 8 桁すべてに数字を埋めるのだ。10 進数を例にとると、われわれなら 12345 と書く数でもコンピュータは 00012345 とする。これがミソだ。実際は、最上位が 0 なら正の数、1 なら負の数と決めている。

そこでコンピュータは、最上位の桁によって正負が区別できることになる。**Java** は 2 進数で加算をしているので、いつか大きな数になったとき最上位が 1 になるときがくる。だから負の数が表示されるのだ。

しかし間違えないでほしい。最上位の 1 は負の数であることを示しているのであって、決して「マイナスの記号の代わりではない」ということなのだ。このことについては、しばらくあとで、別のプログラムに関連して話すことにしよう。