7.2 計算できる循環節

[CycleDec.java] で循環小数の循環節を無駄なく見ることができるようになった。しかし、不十分だ。われわれが知りたいのは、$142857 \times 2 = 285714$ のように循環節が巡回をしているかどうか、である。[CycleDec.java] は、循環節を眺めることはできても計算をすることができない。計算をするためには、商である数字の列が整数値でどこかに格納されている必要がある。

変数 long n でも用意すればよさそうだが、それでは循環節が 20 桁を超えたら対処できない。やはり配列を利用した方がよいだろう。プログラムでは、一つの配列変数に 1 桁の数を与えている。少々無駄が多い—または贅沢なメモリの使い方をしている—と感じるかもしれないね。でも、あとのことを考えれば特別贅沢ということではないのだ。プログラムでは、[CycleDec.java] 同様に循環節を表示することしかしていない。そのため出力は似たようなものに見えるだろう。しかし、今回の数字の列は計算ができるんだ。

programming list [ParaCalc.java]

```
 1: import java.util.Scanner;
 2:
 3: public class ParaCalc {
 4:
 5:     static int[] p = new int[1000];
 6:
 7:     public static void main(String[] args) {
 8:         Scanner s = new Scanner(System.in);
 9:
10:         System.out.print("input 'b' of 1/b: ");
11:         int b = s.nextInt();
12:
13:         if((b % 2 == 0) || (b % 5 == 0)) {
14:             System.out.println("this 'b' is invalid.");
15:         } else {
16:             int r = 1, cyc = 0;
17:             do {
18:                 p[cyc++] = (r * 10) / b;
19:                 r = (r * 10) % b;
20:             } while(r != 1);
21:             disp(cyc);
22:         }
23:     }
24:
25:     public static void disp(int c) {
```

```
26:         int i = 0;
27:         while(c-- > 0) {
28:             System.out.print(p[i++]);
29:         }
30:         System.out.println();
31:     }
32: }
```

プログラムはいきなり 5:行目の `static int[] p = new int[1000];` から始まっている。変数は宣言される場所でスコープ—見渡せる範囲—が異なる。

メソッド内部で宣言された変数は、メソッド内部でのみ有効な変数になる。プログラムが動いているときは、大抵そのとき必要なメソッドが呼ばれている。そして、別のメソッドが必要になれば、一旦いまのメソッドは用なしになる。このとき同時に変数も用なしになる。この仕組みのお陰で、同じ変数名であっても、使うメソッドが違えば平気で使ってかまわないのである。メソッド内で部分的に有効な変数なので**ローカル変数**と呼んだりする。

一方、メソッドの外部で宣言された変数は、内部宣言された変数に較べて "視野" が広い。すなわち広いスコープを持つ。外部宣言された変数はどのメソッドからも参照できる変数ということになる。だから、どのメソッドが使われていようとも、変数は用なしにならない。メソッド全体で有効な変数なので**グローバル変数**と呼んだりする。今回のプログラムは、いくつかのメソッドで配列の値を共有したいので、それらをメソッド外部で宣言した次第である。

では `main()` メソッドを見よう。

ここではたくさんの変数が使われている。`b` は入力される分母の値、`r` は余りが代入される変数だ。`cyc` は循環節が繰り返される回数を代入するためにあるが、`[CycleDec.java]` と違うのは、メソッドにしてあらかじめ調べるのではなく、割り算と同時に調査することだ。配列変数は 5:行目において `int[1000]` で用意したが、ここには 1 桁の数しか代入されない。`int` 型に 1 桁の数しか与えないのは、随分もったいない気がしないでもない。しかし、あとでこの配列は掛け算に利用するつもりでいる。一応 `int[1000]` だから、循環節を 1000 桁持つ分数まで対応が可能だ。その場合、999 までの数が掛けられる可能性があるから、`int` 型が無駄になることはないのだ。

無駄にならないと言っても、ある桁が 9 のときは 999 を掛けても 8991 で頭打ちだ。`int` 型は 4 バイトだから無駄が多すぎる。そう思ったら配列変数だけは `short[1000];` とするとよいだろう。おそらく無駄が減って、メモリの貯蓄ができるに違いない。

13:行目は [CycleDec.java] と同じである。分母に 2, 5 を約数に持つ分数を除外している。

また、17:–20:行目にかけて割り算の商を求めるのだが、ここでは単に商を System.out.print; 文で表示するのでなく、18:行目にあるように、逐一配列変数に代入している。複合的記述で 1 行にしてあるが、本来は p[cyc] = (r * 10) / b;, cyc++; の 2 行で書く文である。cyc は始め 0 であったので、p[0] に最初の商が代入され、cyc++ のお陰で p[1], p[2], ... へと商が代入されるのだ。結局、while 構文の条件により、余りが 1 になるまで繰り返される。これで配列 p[] には各商が代入され、cyc には循環回数が代入されることになった。一石二鳥とはこのことだ。しかも商が配列に格納されたことで、商が数字の羅列でなく、計算できる "数" に昇格したのである。

早速、格納した循環節を表示させよう。そのために作ったのが disp() メソッドで、使われるところは 21:行目である。

disp() メソッドは循環節の表示のために、たとえば 1/7 の循環節であれば、p[0], ..., p[5] の 6 個の値を受け取らなくてはならない。このことを真に受ければ、disp() メソッドは disp(p[0], ..., p[5]) としなくてはならないだろう。しかし、そんなことをしたら disp() メソッドで 1/113 の循環節を表示するには ...。うわー、考えただけでも恐ろしい。

だが、安心してほしい。配列のすべての値をメソッドに渡すには、配列がいくつあるかを知らせればよい。いまの場合、配列の数は cyc で数えているので、disp(cyc) とするだけで、配列の値をメソッドに引き渡すことができるのだ。なぜなら、配列が用意されると、p[0], p[1], p[2], ... という配列は、メモリ上にきちんと順番に割り当てられるからである。その管理はアドレスが担っているのだが、先頭の配列の格納場所さえ分かれば、それ以降の配列の格納場所が特定できるのである。このことは裏を返せば、配列は整然と順番に並べなくてはならないので、int[] p = int[1000]; のように最初に配列の大きさを指定する必要がある。とりあえず int[n] とでもして、あとで n の値を調整するような器用な真似はできないのだ。どうしてもそのようなことがしたいのなら、別のアプローチが用意されているけどね。

ところで、循環節の表示は int[1000] のすべてではない。循環が一巡したところまででよいのだから、繰り返しの回数 cyc 分で十分だ。その意味でも disp(cyc) は十分な仕事をしていることになる。

TRY! このプログラムで、いろいろな分数の循環節を表示させよ。