2.5 よく知られたべき乗計算

豆旅のレベルが 3 になれば、数学ではもっとも有名な数の話題になると予想できるだろう。それにも関係するので、この景勝地では 2^x 以外のべき乗の計算で伏線を張っておきたい。少し唐突だけれど

$$\left(1+\frac{1}{n}\right)^n$$

の値を考えてみよう。$n=1$ のとき、この値は 2 である。$n=2$ なら 2.25 だが $n=3$ から先の計算は厄介である。さあ、コンピュータの出番だ。

programming list [Exp.java]

```
 1: public class Exp {
 2:
 3:     final static int SUP = 100;
 4:
 5:     public static void main(String[] args) {
 6:         System.out.println("EXP(" + SUP + ") = " + exp());
 7:     }
 8:
 9:     public static double exp() {
10:         int i = SUP;
11:         double e = 1.0;
12:
13:         while(i-- > 0) {
14:             e = e * (1 + 1. / SUP);
15:         }
16:         return e;
17:     }
18: }
```

プログラムを見て気付くことは、exp() メソッドが用意されていることと、肝心の main() メソッドがたったの 1 行しかないことだろう。main() メソッドは 1 行しかないが、目的とする計算結果は十分与えくれる。はじめに main() メソッドから見ていこう。

6:行目の System.out.println; 文によって、このプログラムが exp() の**戻り値**を表示することが見て取れる。画面には "EXP(SUP) = exp()" の形式で結果が表示されるはずだ。問題は exp() メソッドが一体何を計算させているかということだ。

9:行目からは、exp() メソッドが浮動小数点数を返すことが分かる。それは exp() が double で

宣言されているので一目瞭然だ。ところで double exp() は () 内に何も記述されていない。これは exp() メソッドがどこからも値を受け取らないからである。値は受け取らないくせに、自分からは何らかの値を返す。何とも身勝手なメソッドではないか。

実はこのメソッドが値を受け取らないのには理由がある。いまは $(1+1/n)^n$ を計算したいのだが、n にあたる値はすでに 3:行目の SUP で決めてしまっているからだ。この場合は、はじめから $(1+1/100)^{100}$ の計算をすることを目的としている。だから、exp() は値を受け取る必要がなかったのである。

それより、これまでと違って SUP が main() メソッドの外側で宣言されているのはなぜだろう。そして、final が変更しない定数を表すのはよいとしても、static が付いているのはなぜだろう。

まず、SUP がメソッドの外側で宣言されているのは、変数には**スコープ**が関わるからだ。スコープとは変数が有効となる範囲である。一般に、メソッド内で宣言された変数はそのメソッド内だけで有効となる。ここでは main() メソッドと exp() メソッドの二つがあり、その両方で SUP が利用される。したがって、どちらか一方だけで SUP を宣言したのでは他方で有効にならない。そこで、二つのメソッドの外で宣言したのだ。このことで SUP のスコープは Exp クラス内全体となる。

また、SUP に static を付けて宣言したのは、static なメソッドである main() と exp() から参照されないからである。static の詳細は述べないけれども、端的に言って static がないとエラーになる。

さて、double exp() メソッドは $(1+1/100)$ を 100 回掛けることになる。そこで 10:行目でカウント用の変数 i に SUP の値を代入している。ここで 3:行目の SUP = 100 の効用に注目してもらいたい。定数値 100 は 6:行目と 10:行目と 14:行目にある。あらかじめ 3:行目で SUP の値を定義したために、今後 SUP の値を変更したくなった場合でも、3:行目だけの変更で済む。これは大事なことだ。いま SUP の値を変更したくなった場合と言ったが、一度 final で宣言されたものは、代入による変更が利かないことに注意してもらいたい。だから 13:行目の while 構文は SUP-- > 0 ではなく i-- > 0 なのだ。このようなことがあるものの、定数のような値は final で宣言するのが望ましい。

exp() メソッドにおいて、変数 e には結果が代入されるが、掛け算の最終結果を求めるには、初期値を 1 としておかなくては具合が悪い。もちろん浮動小数点数を扱うので 11:行目では 1.0 としている。

13:–15:行目の手法は `[XpowerOf2.java]` のときと同じだ。これで 100 回分の掛け算が行われる。また、14:行目の `1.` の表記は間違いではない。**Java** は小数点さえあれば、その数を浮動小数点数として扱う。したがって `1.` は `1.0` のことである。それなら、もう一ケ所ある `1` や `SUP` の値に小数点は不要なのだろうか。結論を言えば、付けるに越したことはないが不要である。なぜなら、浮動小数点数と整数の演算では精度の高い型が採用されるからである。つまり 14:行目の一連の演算は、どの段階を取っても浮動小数点数で計算されている。

16:行目で返される `e` の値、すなわち `exp()` メソッドが返す浮動小数点数によって、6:行目の `System.out.println;` 文が滞りなく行われる寸法である。

さて、結果はどうなっただろうか。

TRY! `SUP = 100` の値を大きくしてみよ。君たちのコンピュータはどのぐらいの大きさまで対応できるだろうか。また、その際に出力される結果がどうなるか確認してみよ。

TRY! はじめから `SUP = 100` で値を決めるのではなく、$(1+1/n)^n$ の n を入力して結果を表示するプログラムにしてみよ。

何回か試してみれば分かるように、プログラムは一定の値になるように感じるだろう。結論を言えば、この計算はある値—もちろん今回の豆旅にふさわしい値だ—に**収束**することが知られている。しかも数学では大変重要な値になっている。これが何の値かは、ずっと先まで豆旅を続ける必要がある。楽しみを先延ばしにして悪いが、いまはこの景勝地から離れることにしよう。