1.3 フィボナッチ数列の 2 項間の比

フィボナッチ数列には面白い性質がある。その一つは、隣り合う 2 項の比を調べることで見えてくる。1, 1 から始まるフィボナッチ数列を 2 項ずつペアにし、

$$\frac{1}{1}, \frac{2}{1}, \frac{3}{2}, \frac{5}{3}, \frac{8}{5}, \frac{13}{8}, \frac{21}{13}, \frac{34}{21}, \cdots$$

のように分数を作る。この程度は暗算や電卓で計算できるだろう。

EX. 実際に比を計算してみよ。

しかし、こういったことを調べるには、やはりコンピュータが適している。比の計算ということは、分数、つまり割り算をしなくてはならない。割り算をすれば、必ずといってよいほど小数の値がでるものだ。しかし、割り算には特有の問題が含まれている。はじめにそれを説明しよう。

programming list [Ffailure.java]

```
1: public class Ffailure {
2:
3:     public static void main(String[] args) {
4:         final int SUP = 20;
5:         int f1 = 1, f0 = 1;
6:
7:         for(int i = 3; i <= SUP; i++) {
8:             System.out.println((f1 + f0) / f1);
9:             f1 = f1 + f0;
10:            f0 = f1 - f0;
11:        }
12:    }
13: }
```

プログラムは [Fibonacc.java] の 8:行目を変えただけである。新しいフィボナッチ数を現在の f1 で割れば、2 項の比になる。ここで改行するよう指示しているのは、桁の多い小数を横に並べるのは見苦しいと感じたからだ。それが見苦しいと感じなければ改行しなくてよもい。

実行すれば分かることだが、電卓で計算したようにならなくてガッカリするだろう。この原因は 8:行目だが、遠因は 5:行目に存在している。8:行目の (f1 + f0) / f1 の計算は、5:行目より整数どうしの割り算になる。まあ、これは当然だ。ところで **Java** は、整数どうしの演算は結果も整数にしてしまうし、浮動小数点数どうしの演算は結果も浮動小数点数になるものと思っている。**Java**

にとっては `8 / 5` の演算結果は `1` であり、たとえ計算結果を `1.6` と認識していても `1.000000` としか表示しないのだ。ところが **Java** はちゃっかりしている。`8.0 / 5.0` の演算なら `1.600000` と表示するのだ。

この性格を見越せば解決は簡単だ。浮動小数点数の答がほしければ、浮動小数点数どうしの割り算にしてやればよいだけだ。`f1` と `f0` は `double` 型にすればよい。

programming list [Fratio.java]

```
1: public class Fratio {
2:
3:     public static void main(String[] args) {
4:         final int SUP = 20;
5:         double f1 = 1.0, f0 = 1.0;
6:
7:         for(int i = 3; i <= SUP; i++) {
8:             System.out.println((f1 + f0) / f1);
9:             f1 = f1 + f0;
10:            f0 = f1 - f0;
11:        }
12:    }
13: }
```

`f1` と `f2` の型を `double` にした際、`f1 = 1.0` のように小数点を付けて代入していることに注意を払ってほしい。5:行目の宣言が `double` であるから、たとえ `double f1 = 1,` であっても 8:行目の計算は浮動小数点数で行ってくれる。しかし `double` で宣言すると同時に浮動小数点数で代入するほうが無難だし、分かりやすい。

プログラムを実行すると、フィボナッチ数列の隣り合う 2 項の比が一定になっていく様子が分かるだろう。この値が何かは、のちの話題となる。

TRY! フィボナッチ数列は `f1 = 1.0, f0 = 1.0` で始めているが、これを違う値で始めると 2 項の比に変化があるか確かめよ。

TRY! 隣り合う 2 項の比でなく、一つおきの 2 項の比を計算すると、比の値はどうなるか？

今回、精度の向上を狙って `double` 型を選んだ。精度が向上することはよいことだが、余計にメモリを食うことも事実だ。どのみちコンピュータでは、無制限に高い精度を得られないのだから、`double` の使い過ぎは無駄遣いに通じる。ここでは `float` 型でも十分である。もっとも近頃では、

そんなにメモリの無駄遣いに気を配ることもない。普段使いの機器で簡単なプログラムを組んでいる豆旅ではなおさらである。通常は `double` 型一辺倒で問題ない。

TRY! 5:行目の `double` を `float` に変えて実行してみよ。精度に違いはあったか。