0.2 無限

さあ整数が出揃って、いよいよ数学の世界の奥へ入っていく準備ができた。数は整数の他にも、様々な種類のものがあることは知っているだろう。続けていろいろな種類の数を紹介したいけれど、まだ整数についてさえ十分に語っていないのだ。それは何かって？　それは "..." の部分。特に "..., −10" と "10, ..." のところは、単なる省略ではないことを言いたいのである。

ここに使われた "..." は、この先の数を書く意味がないことを示している。なぜなら、書き切ることができないから。当たり前だよね。ところが "..." には、思いもかけない不思議が詰まっている。

君たちは**偶数**と**奇数**は知っているね。自然数に限れば 2, 4, 6, ... が偶数、1, 3, 5, ... が奇数ということになる。ものの集まりを**集合**と呼ぶことにして、偶数の集合を $\mathcal{E}$、奇数の集合を $\mathcal{O}$、そして自然数の集合を $\mathcal{N}$ で表せば

$$\mathcal{E} + \mathcal{O} = \mathcal{N} \tag{1}$$

であることが想像できる。でも、残念ながら違う。想像が事実に合わないことはよくある話。(1) は偶数や自然数のような、**無限集合**にはそぐわない関係なのだ。それについてカントール[1)]は次のように説明している。

たとえば "りんごが 5 個ある" とは、各りんごに自然数 1, 2, 3, 4, 5 が**一対一に対応**していると見る。このとき、りんごの数 (かず) は 5 に等しいと言える。同様のことを偶数と自然数で考えてみよう。すると

$$\begin{array}{lccccccc} \text{偶数} & 2 & 4 & 6 & 8 & 10 & \dots \\ & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \cdots \\ \text{自然数} & 1 & 2 & 3 & 4 & 5 & \dots \end{array}$$

のように、各偶数に自然数 1, 2, 3, 4, 5, ... が一対一に対応していることになる。したがって偶数と自然数は同じ数 (かず) だけある、と考えるわけだ。このときに数 (かず) ということばは適切でないように思えるので、カントールは**濃度**という用語を使っている。すると奇数に対しても、奇数と自然数は同じ濃度を持つと言えるのだ。

この用語はまったくもって適切で、われわれの生活感覚にも合っている。なぜなら、同じ濃度の

1)　ゲオルク・カントール (1845–1918)：ドイツの数学者。

食塩水を混ぜると、同じ濃度を持つ食塩水ができるのだから。そして食塩水の "量" も増えているので、自然数も偶数だけの "量" に比べると、きっと奇数の分だけ "量" が増えているんだろう。

ちょっと不思議な感覚だけど、日常からかけ離れたものにはありがちなことではある。(想像を超える大金持ちの資産) + (想像を超える大金持ちの資産) は、やはり (想像を超える大金持ちの資産) であるのと同様、"..." はわれわれの想像を超えた世界ということなのだ。

それでは、コンピュータは無限にどう対処しているのだろう？　ここで簡単な調査をしておこう。

programming list [Parrot.java]

```
 1: import java.util.Scanner;
 2:
 3: public class Parrot {
 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 n = s.nextInt();
10:
11:         System.out.println("an input number is " + n);
12:     }
13: }
```

このプログラムは、実行後に変数 n に**任意の**数値を代入できることを確認するだけのもので、前節の [PrintOut.java] とさして変わらない。[PrintOut.java] では、変数 n への代入はプログラム中で行っていたことを思い出そう。でも、なんだかプログラムがにぎやかになっている。プログラムを説明する前に、これをテキストファイル Parrot.java に保存し

```
% javac Parrot.java
```

を実行して Parrot.class ファイルを作ったあと

```
% java Parrot
```

としてみよう。画面に "input a number: " が出力されただろう。つまり **Java** が数値の入力を促（うなが）しているのだ。ここでたとえば 123 を入力すれば、"an input number is 123" が表示される。動作自体は [PrintOut.java] と同じである。

プログラムを簡単に説明しておこう。1:行目に import java.util.Scanner; が書かれているね。

Java は特別なことをする場合、そのために必要な道具—**クラスライブラリ**という—をその都度調達する必要がある。ここではキーボードからの入力を受け付けたいので、java.util.Scanner（java.util パッケージに属する Scanner クラス）が必要になったのだ。前節の最後に例示したプログラムにも import 文があっただろう。あれは、グラフィックウィンドウに表示させるアプレットのプログラムだったので、それ用のクラスライブラリを必要としたのだ。

で、Scanner クラスは 6:行目のように書いて使う。豆旅の出発直後にこの説明は厳しいのだが、ここではとりあえず変数の代入と対比させて

```
int n          =          456;
整数変数 n を   初期化する   456 で
Scanner s      = new      Scanner(System.in);
スキャナ変数 s を 初期化する  スキャナメソッド System.in で
```

のように読み替えてみよう。だいぶ変な言い回しだけど、要するに読み込み機能のための変数—この場合は**インスタンス**という—を用意している。そのため 9:行目にあるように、s に持たせた機能 nextInt() によってキーボードから読み込んだ値を変数 n へ代入できるのである。ただし、nextInt() の機能はただ入力を受けるだけなので、画面に何も表示されない。そこで注意を引くため 8:行目でプロンプトとして “input a number: ” を表示させているのである。nextInt() を用いる場合は直前に、何をするか分かるようなメッセージを表示させよう。

11:行目は [PrintOut.java] と同じく、結果の表示である。

さて、結果が表示されたらプログラムは終了するのだが、ここで大事なことを述べておこう。本来なら、最後に s.close(); を記述してインスタンスを**解放**するところである。そのためには**例外処理**が必要となる。しかし、それを丁寧に記述していると豆旅の肝心の見所がかすんでしまいかねない。インスタンスは単に、豆旅をちょっとだけ快適にするために生成したのであり、この程度のプログラムでは s.close(); がなくても問題は起きないであろう。この先も例外処理は省くので、危ない道を通ることもあると承知してほしい。本当に危険な道に入りそうなら.close(); を使おう。ただし、使い方はインターネットで調べてもらうことになるけれど。

では、いろいろな数を入力してもらいたい。いつでも納得できる結果になっただろうか？

TRY! [Parrot.java] はどれだけ大きい整数に対応できるだろうか。また、小数値を与えるとどうなるだろうか。

試して分かっただろうが、残念なことに極端に大きい数はエラーを引き起こす。残念ながら無限

に大きい数は扱えない。現実的には無限を使うことはないけれど、数学においてはそれでは困るのである。また、小数値も受け付けなかったはずだ。これはさらに困ったことだ。豆旅を続けるには、まずそれらを解決しなければならない。