0 . . . の豆旅

0.1 自然数から整数への継承

われわれはものを数えるとき

$$1,\ 2,\ 3,\ 4,\ 5,\ \ldots,\ 10,\ \ldots,\ 100,\ \ldots \tag{1}$$

と数えるのが普通である。これらの数はどこまでも大きく数えられるし、どこまでいっても終わりというものがない。数学の世界では、ここに登場した数を**自然数**と呼んでいる。

まず、約束がなされたことに気づいただろうか。(1) の数を自然数と呼ぶ約束のことだ。ここで、何でそう呼ぶの？とか、0 は自然数じゃないの？とかの疑問が浮かぶかもしれないね。数学では "何々のことを何々と決める" ということが頻繁にでてくる。そしてその決め方が自然と納得できるものもあれば、場合によってはどうしてそう決めるのか不思議に思うことがあるだろう。約束—すなわち**定義**—というものは、大体の雰囲気で決める場合もあれば、深い意味があってそう決める場合がある。君たちが戸惑うのは、深い意味があって定義されることがらだろう。その場合は、定義に納得いかないこともあるはずだが、深い意味があるだけに当面は謎のままになってしまうものだ。その定義に関する内容を深く理解したとき、謎が氷解することが往々にしてあるので、楽しみはとっておくのがよいだろう。

自然数という呼び名は、大体の雰囲気から妥当と思われる。人間が自然発生的に使い出した数が 1, 2, 3, . . . だから。

ちょっと待ってほしい。われわれが現在、自然に使う数字は 0, 1, 2, 3, . . . ではないか。それなら 0 を含めて自然数と呼ぶ方が自然だろう。こう考える人はいるかな。たしかに、そのとおり。それは間違った考えではない。ただ、ここでは習慣に従い、(1) が自然数であると定義しておこう。あとで分かるように、0 は特別扱いしたい数だから。

ところでわれわれが普段使う数には -1, -5 などもある。このような数は 0 より小さい数を表すために作られた数だ。自然界には 0 より小さいものはないと言ってよいだろう。何もない状態が 0 の状態だから、もうこれ以上なくなる状態はありえないのだ。ところが人々は 0 より小さい状態を

概念として作ってしまった。借金や気温や水深などはそのよい例になっている。

$$\begin{aligned} 5{,}000\text{ 円の借金} &\quad\to\quad -5000\text{ 円} \\ \text{氷点下 }8\text{℃} &\quad\to\quad -8\text{℃} \\ \text{水深 }120\,\text{m} &\quad\to\quad -120\,\text{m} \end{aligned}$$

0 より小さい数は、自然数に − の**符号**をつけて表している。1 や 5 は、0 より 1 や 5 だけ大きい数ととらえるならば、−1 や −5 は、0 より 1 や 5 だけ小さい数ととらえることができる。数に − の符号をつけて、0 より小さい数を表すことは新たな約束となる。しかし、ここでは約束もさることながら、数の継承がなされたことに注意を払ってほしいのだ。

もし、負の数を新たに定義するだけなら a, b, c, ... という "数" を作れば済む。ところが実際は、自然数を土台に −1, −2, −3, ... と数を拡張している。つまり自然数を継承したわけだ。継承するということは、自然数の持つ性質も受け継ぐことを意味している。たとえば数の間隔は、負の数でも 1 ずつだし、大きい数字を使うほど 0 から離れた数になることなどがそうだ。

結局、数は 0 を中心にして

$$\ldots,\ -10,\ \ldots,\ -3,\ -2,\ -1,\ 0,\ 1,\ 2,\ 3,\ \ldots,\ 10,\ \ldots$$

と並んでいることになる。そしてこれらの数を**整数**と呼び、さまざまな数の基準として使うことにするわけである。

それではこの辺で、**Java** に何かさせてみよう。

programming list [PrintOut.java]

```
1: public class PrintOut {
2:
3:     public static void main(String[] args) {
4:         int n;
5:         n = 123;
6:         System.out.println("a number is " + n);
7:     }
8: }
```

この旅では規模の小さな豆粒並みのプログラムしか書かない。だから、OS 付属のテキストエディタで十分いける。このコードをファイル名 "PrintOut.java" でテキストファイルに保存しよう。そして、ファイルを保存したディレクトリで Terminal を起動して、プロンプト % に対して

```
% javac PrintOut.java
```

と入力し return キー（Enter キー）を押す[1)]。テキストに書き間違いなどなければ何事もなかったように、プロンプトだけが表示されるはずだ。そこで今度は

```
% java PrintOut
```

を実行してもらいたい。javac とタイプしたときは**拡張子**.java を含めて入力したが、java とタイプするときは拡張子を付けていないことに注意しよう。すると画面には実行結果の

```
a number is 123
```

が表示される。うまくいっただろうか。

この豆旅は、数学の話題にはなるべく詳しい説明をする予定だが、**Java** のコードについてはいまみたいにあっさり通過する。理由は、数学の考え方を重視しているからだ。もしプログラミングに関する一連の流れが難しいと感じるなら、それは言語の理解が不足しているためではない。問題解決のための知識が不足しているのだ。**Java** 自体を詳しく知りたければ、何か別の書物か **Java** に関するサイトを調べてもらいたい。それは自分で解決することなのだ。では、プログラムの説明に入ろう。

1:行目の `public class PrintOut {` は、これが `PrintOut` という名のクラスファイルであることを意味し、この行の `{` から 8:行目の `}` までが一つのグループを形成している。実際、`% javac PrintOut.java` を実行すると同じディレクトリに `PrintOut.class` という名のファイルができる。そのことは、Terminal なら `ls` コマンドで確認できる[2)]。

さて、このプログラムは画面に "a number is 123" と表示し、**復帰改行**—改行してカーソル位置を行の先頭に移動—するだけのものである。そのことを記述した本体が 3:行目から 7:行目にかけて書かれた**文**だ。この文も `{` と `}` の間を一つのグループにしている。ここでは 4:, 5:, 6:行の命令がひとまとめにされていて、そのグループの名前が `main()` であるととらえてもらえばよい。

`main()` の前についているキーワード `public` は `main(){}` が『公開されている』ことを意味し、キーワード `static` は `main(){}` が『静的である』ことを意味し、キーワード `void` は `main(){}` が『値を返さない』ことを意味している。と言っても、いまの段階では何のことを言ってるのかさっぱりだろう。また、`()` 内の `String[] args` は**引数（ひきすう）**をとる場合の記述だが、これらの説明は豆旅の行程が進んでからするので、当面はこう書くのだと納得してもらうしかない。それよ

1） MacOS のターミナルの場合。Windows ならコマンドプロンプトを起動して同様に入力する。
2） Windows のコマンドプロンプトなら `dir` コマンドを使う。

り肝心の本体の説明に入ろう。

まず 4:行目に `int n;` がある。これは**変数** `n` を整数値で扱うことを**宣言**している。`int` は次に続く変数が、整数値であることを明示するためのキーワードになっている。したがって `n` が `1.23` のような小数になることはない。小数を扱うにはそのためのキーワードで指示する必要があるが、それはあとで登場する。いずれにせよ変数を使うには、その変数が整数なのか小数なのかを明らかにしなくてはならない。`int` は `n` が整数**型**であることを **Java** に知らせるキーワードなのだ。また `;` は文の終わりを示す記号で、この行には必要である。うっかり忘れると **Java** はエラーを返してくる。プログラムは多くの文が集まってできているので、大抵の命令の末尾には `;` が付くものだと考えておいてほしい。

5:行目には `n = 123;` がある。多くのプログラミング言語では、`=` は左辺と右辺が等しいことを意味しない。右辺の値を左辺の変数に**代入**することを意味する。したがって、ここでは `n` に `123` を代入している。数学的には代入というが、**Java** 的には変数 `n` を `123` で**初期化**するという。初期化の時点で `n` は `123` の値を持ったことになる。この行も文であるから、当然のごとく末尾に `;` が付いている。

さあ、6:行目に行こう。こいつは少々厄介だ。`System.out.println();` は `()` 内の**書式**に従って出力—画面に表示—するよう指示する命令である。まず、`""` で囲まれた内容はそのまま出力される。そのため **Java** は画面に、“a number is ” と出力する。続く `+` は足し算の意味で使われていない。`System.out.println();` 内においては、`+` は「文字の連結」を意味することがある。だからといって “a number is ” に続いて “n” の文字が出力されるわけではない。`n` は `""` で囲まれていないので、`n` が持つ “値” を出力するのだ。`n` には `123` が代入されていたはずだから、`n` の部分は `123` に取って代わって出力されることになる。つまりコンピュータは画面に “a number is 123” を表示し、カーソルは次の行の先頭に移るのである。

このように、ある部分ではプログラミングは易しいものだが、ある部分では非常に気難しい面もある。この豆旅では気難しいことは省いておこう。まあ、それでも多少のプログラミング作法は身に付くだろうから。それに何が気難しいかは、ほんの一握りの好奇心があれば誰にでも体験できることだ。たとえばいまの例では、`n = 123;` で代入する値をいろいろ変えて試すことで、多少の好奇心は満たされるだろう。だが、好奇心は満たされても不満が募るかもしれない。不満の解消はしばらく待ってもらうしかないが。

TRY! 5:行目の `n = 123;` を `n = -123;` や `n = 1.23;` に変えて実行するとどんなことが起きるか？

TRY! `System.out.println();` の () 内を、`n * 3` や `n / 100` に変えて実行するとどうなるか？

TRY! `System.out.println();` の () 内全部を、`n + n + "a"` や `"a" + n + n` や `"a" + ( n + n )` に変えて実行するとどうなるか？　これで + の振る舞いが想像できるだろうか？

ところでいま実行したプログラムは、コンソールウィンドウと呼ばれる画面に文字を出力している。**Java** らしく普通のウィンドウに表示させたいときは、たとえば

```
import java.applet.Applet;
import java.awt.Graphics;
public class PaintOut extends Applet
{
    public void paint( Graphics g ) {
        int n = 456;
        g.drawString( "a number is " + n, 25, 25 );
    }
}
```

のような記述をするものだが、この豆旅のプログラムは数値を出力することしかしない。よって、これからもコンソールウィンドウに出力するようにコードを書くことにする。実際に **Java** のアプリやブラウザに表示する見栄えのよいプログラムを作りたければ、それなりの開発環境で **Java** の書物を参考に研究してもらいたい。この豆旅は、単に **Java** のコードを使って、ほんの少し数学に浸るだけのものだからだ。

ところで、上に示したコードの 6:行目では `int n = 456;` と、変数 `n` を宣言すると同時に `456` を代入している。**Java** ではこのような記述ができるので、今後もこの手の書き方は多用するつもりである。慣れてしまえば何のことはないだろう。