情報科学特殊講義'2000 ＃2

久野　靖*

1999.1.24-27

0　はじめに

Java 初体験はいかがでしたか? 2 回目ではまず、1 回目の例題のいくつかについて解説しながら、いくつか注意しておきたい点を説明します。続いて 1 回目でちょっとだけ学んだ「オブジェクト」の利用について、文字列オブジェクトを題材として演習していきます。

1　字下げその他について

演習 6a や 6c は 6b ができれば同じことだから略。6b の疑似コードは次の通り。

- 実数 x、y を入力する。
- x+y、x-y、x*y、x/y を出力する。

で、Java では次のようになる。

```
import java.io.*;

public class r1ex6b {
  public static void main(String args[]) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("x = "); System.out.flush();
    double x = (new Double(in.readLine())).doubleValue();
    System.out.print("y = "); System.out.flush();
    double y = (new Double(in.readLine())).doubleValue();
    System.out.println("x + y = " + (x+y));
    System.out.println("x - y = " + (x-y));
    System.out.println("x * y = " + (x*y));
    System.out.println("x / y = " + (x/y));
  }
}
```

自分が作ったのと違う、と思いました? もちろん、それで当然であり、どっちが正解ということはない。ちょうど日本語で同じことを言うのにさまざまな言い方があるのと同じこと。ただし、スマートだとか短いとか分かりやすいとかそういった点で違いはあるかもしれない。美観の問題! だから自分の好みもとりまぜて、考えて選ぶこと。

その他注意してほしい点:

- まず、字下げ (左側に空白を入れること) をしない人がいるけど、これは絶対やめた方がよい。たとえば、

*筑波大学大学院経営システム科学専攻

```
if(...) {          if(...) {
    ....               ....
    ....               ....
}                  }
```

左のように書いてあれば、どこまでが「もし〜ならば」の範囲かすぐ分かる。しかし右のようにべったり揃えてしまうとそれが分からないし、たとえば間違って「}」をつけ忘れたり別の場所に入れてしまうともう大混乱になる。たかがスペースキーを打つ手間を惜しんで後で間違い探しに 1 時間も掛けるのは阿呆みたいでしょ? プログラムは「美しく」書こう。

- 和、差、商、積をいったん変数に入れていない。もちろん、前回の例題のように変数に入れてもよいが、上のようにいきなり出力してもよい。一般に変数が書けるところには代りに任意の計算式を書いてもよい。ところで、式を丸かっこで囲んでいるのは、「`"...." + x + y`」とすると足し算は行われず全部文字列として連結されてしまうから。

2 枝分かれについて

演習 7a は 2 つの枝分かれですから例題とほとんど同じ。まず疑似コードを見よう。

- 実数 a、b を入力する。
- もし a > b であれば、
- 　 a を出力する。
- そうでなければ、
- 　 b を出力する。
- 枝分かれおわり。

Java では次の通り。

```
import java.io.*;

public class r1ex7a {
  public static void main(String args[]) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("a = "); System.out.flush();
    double a = (new Double(in.readLine())).doubleValue();
    System.out.print("b = "); System.out.flush();
    double b = (new Double(in.readLine())).doubleValue();
    if(a > b) {
      System.out.println("larger : " + a);
    } else {
      System.out.println("larger : " + b);
    }
  }
}
```

しかし、次のような「別解」はどうだろう。

- 実数 a、b を入力する。
- max ← a
- もし b > max であれば、
- 　 max ← b

- 枝分かれおわり。
- max を出力する。

これの Java 版は次のとおり。

```
import java.io.*;

public class r1ex7a1 {
  public static void main(String args[]) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("a = "); System.out.flush();
    double a = (new Double(in.readLine())).doubleValue();
    System.out.print("b = "); System.out.flush();
    double b = (new Double(in.readLine())).doubleValue();
    double max = a;
    if(b > max) {
      max = b;
    }
    System.out.println("larger : " + max);
  }
}
```

どっちが好みですか? これもどちらが正解ということではない。

しかし 7b はもうちょっとややこしい。まず考えるのは、a と b の大きい方はどっちか決めて、それぞれの場合についてそれを c と比べるというもの。

- 実数 a、b、c を入力する。
- もし a > b であれば、
- 　もし a > c であれば、
- 　　a を出力する。
- 　そうでなければ、
- 　　c を出力する。
- 　枝分かれおわり。
- そうでなければ、
- 　もし b > c であれば、
- 　　b を出力する。
- 　そうでなければ、
- 　　c を出力する。
- 　枝分かれおわり。
- 枝分かれおわり。

うーむ大変だ。これを Java にしておく。

```
import java.io.*;

public class r1ex7b {
  public static void main(String args[]) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("a = "); System.out.flush();
    double a = (new Double(in.readLine())).doubleValue();
    System.out.print("b = "); System.out.flush();
    double b = (new Double(in.readLine())).doubleValue();
```

```
    System.out.print("c = "); System.out.flush();
    double c = (new Double(in.readLine())).doubleValue();
    if(a > b) {
      if(a > c) {
        System.out.println("largest : " + a);
      } else {
        System.out.println("largest : " + c);
      }
    } else {
      if(b > c) {
        System.out.println("largest : " + b);
      } else {
        System.out.println("largest : " + c);
      }
    }
  }
}
```

こうなると字下げしてないとごちゃごちゃになるでしょう? しかし字下げしてあってもこれはかなり苦しい。ときに、先の別解から発展させるとどうだろう?

- 実数 a、b、c を入力する。
- max ← a
- もし b > max であれば、
- 　　max ← b
- 枝分かれおわり。
- もし c > max であれば、
- 　　max ← c
- 枝分かれおわり。
- max を出力する。

この方がすっきりしているでしょう? Java でも次のとおり。

```
import java.io.*;

public class r1ex7b1 {
  public static void main(String args[]) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("a = "); System.out.flush();
    double a = (new Double(in.readLine())).doubleValue();
    System.out.print("b = "); System.out.flush();
    double b = (new Double(in.readLine())).doubleValue();
    System.out.print("c = "); System.out.flush();
    double c = (new Double(in.readLine())).doubleValue();
    double max = a;
    if(b > max) { max = b; }
    if(c > max) { max = c; }
    System.out.println("largest : " + max);
  }
}
```

今度はどちらが好みですか? 一般には、枝分かれの中に枝分かれを入れるよりは、枝分かれを並べるだけで済ませられればその方が分かりやすいといえる。なお、if 文が 1 行に書かれているが、どこで行を変えるかは自由に選べるので、この場合はこの方が見やすいと思ってこうしてみた。

3 多方向の枝分かれ

演習 7c は 3 通りに分かれるのだから、if の中にまた if が入るのはやむを得ない。しかし、ちょっと工夫すると分かりやすくなる。Java コードから見ていただく。

```
import java.io.*;

public class r1ex7c {
  public static void main(String args[]) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("x = "); System.out.flush();
    double x = (new Double(in.readLine())).doubleValue();
    if(x > 0.0) {
      System.out.println("positive.");
    } else if(x < 0.0) {
      System.out.println("negative.");
    } else {
      System.out.println("zero.");
    }
  }
}
```

これは実は最初の if の else のすぐ後ろに次の if がくっついた、という形をしているが、こういうパターンを利用するとプログラムが分かりやすくなる。順序が前後したが、疑似コードだと次のようになる。

- 実数 x を入力する。
- もし $x > 0$ ならば、
- 　　「positive.」と出力。
- そうでなくて $x < 0$ ならば、
- 　　「negative.」と出力。
- そうでなければ、
- 　　「zero.」と出力。
- 枝分かれおわり。

一般に「そうでなくて〜ならば、」は何回現われてもよい。またそのどれもが成り立たない場合は「そうでなければ」に来るが、この部分は不要ならなくてもよい。これを Java にする場合は次の形になる。

```
if(...) {
  ...
} else if(...) {
  ...
} else if(...) {
  ...
} else {
  ...
}
```

なお、これはあくまでも Java の if 文の「else の後にすぐ次の if をくっつけた」というパターンなだけで、特別な文というわけではない。

4　繰り返しについて

演習 8a であるが、これは前回説明したように 2 ずつ引いていけばよい。

- 整数 x を入力する。
- x > 1 である間繰り返し:
- 　　x ← x − 2
- ここまで繰り返し。
- もし x = 0 ならば、
- 　　「偶数」と出力。
- そうでなければ、
- 　　「奇数」と出力。
- 枝分かれ終わり。

Java で書くと次の通り。

```
import java.io.*;

public class r1ex8a {
  public static void main(String args[]) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("x = "); System.out.flush();
    int x = (new Integer(in.readLine())).intValue();
    while(x > 1) { x = x - 2; }
    if(x == 0) {
      System.out.println("even.");
    } else {
      System.out.println("odd.");
    }
  }
}
```

このような繰り返しの条件に注意。「x > 1 の間繰り返す」という条件で、しかも繰り返しの中では x は小さくなる方向に変化するから、いつかは x が十分小さくなってこの繰り返しは確実に止まる。さらに、繰り返しが止まったときは「$x \leq 1$」が成り立っているから、つまり x は 1 であるか 0 であるかのいずれか、になっている。なのでそのどちらかを調べれば奇数か偶数かが分かる。つまり、繰り返しを使う時には

- 最終的に成り立って欲しい条件を決め、
- 繰り返しの内側ではその条件が成り立ちやすい方向に変化を起こす

ようにする。これらが守られていないと、繰り返しが無限に続いたり、繰り返しが終わった時に役に立つ値が計算されていない、ということになる。

たとえば、人生で金持ちになるための手順も同様である。

- 手持ちの金額が G 円未満である間繰り返し、
- 　　手持ちの金を増やす※。
- ここまで繰り返し。

ただし問題は、人生では※を確実に行う手順が知られていないことである。

ところで、x が正の整数でないと、当然このプログラムは正しく動作しない。それでいいのか、と思うかも知れないが、それは構わない。プログラムを作る時はあくまでも「プログラムが動作すべき条件が満たされた時に正しく動作するプログラムを作る」ことだけが求められる。余計なオマケのために労力を費すのがいいかどうかは必ずしも分からない。

ただし、自分でプログラムの動作を定める時には、正の整数以外でも扱いたいかどうか、十分考えてよいと思うように決めること。

5　for 文による繰り返しについて

演習 8c は 8b が分かれば同じこと。8b は要するに y を x 回足せばよい。これを行うのにいろいろな方法があるが、普通はもう 1 つ変数を用意して、これを 1 ずつ足しながら回数を数えていく。

- 整数 x、y を入力する。
- seki ← 0。
- i ← 0。
- i < x である間繰り返し:
- 　　seki ← seki + y。
- 　　i ← i + 1。
- ここまで繰り返し。
- seki を出力する。

これを Java にすると次の通り。

```
import java.io.*;

public class r1ex8b {
  public static void main(String args[]) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("x = "); System.out.flush();
    int x = (new Integer(in.readLine())).intValue();
    System.out.print("y = "); System.out.flush();
    int y = (new Integer(in.readLine())).intValue();
    int seki = 0;
    int i = 0;
    while(i < x) {
      seki = seki + y;
      i = i + 1;
    }
    System.out.println("product is: " + seki);
  }
}
```

ここでもいくつか注意を。

- 上のコードでは「x 回繰り返す」のに、i を 0、1、2、…、$x-1$ まで変化させている。このように「0 から数える」のは計算機ではよく使う。
- まず、変数 seki を最初 0 にしておき、次に y ずつ増やしていることに注意。合計を求めるには「`seki = seki + y`」のようにして「足し込んで」行くのが定石。なお、これはよくでて来るパターンなので「`seki += y`」と書くこともできる。他に「引き込む」「掛け込む」等もあり。
- 特に「1 足す」「1 引く」はよく使うのでさらに短く「`++i`」「`--i`」と書くこともできる。

さて、上で出て来た i のように「数を数える」ための変数を「カウンタ」と呼ぶ。カウンタを 0、1、2、…、$N-1$ まで変化させて繰り返す、というのはとてもよく使うので疑似コードで次のように書くことにする。

- 変数 i を 0 から $N-1$ まで変化させながら繰り返し:
- 　　....
- 繰り返し終わり。

また、Java でもこの部分は while 文の代りに for 文を使って書くことが多い。

```
for(int i = 0; i < n; ++i) {
    ....
}
```

for 文は while 文の「親戚」だが、カウンタ用の変数の初期設定と毎回の更新 (加算) を一緒に書いて見やすくできる。なお、上のようにカウンタ変数の宣言を for の中に入れた場合は、その変数は for の内部でのみ使える。これを使って先の例題を書き直すと次の通り。

- 整数 x、y を入力する。
- seki ← 0。
- 変数 i を 0 から x-1 まで変化させながら繰り返し:
- 　　seki ← seki + y。
- ここまで繰り返し。
- seki を出力する。

この方が見やすいでしょう? Java では次の通り。

```
import java.io.*;

public class r1ex8b1 {
  public static void main(String args[]) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("x = "); System.out.flush();
    int x = (new Integer(in.readLine())).intValue();
    System.out.print("y = "); System.out.flush();
    int y = (new Integer(in.readLine())).intValue();
    int seki = 0;
    for(int i = 0; i < x; ++i) {
      seki = seki + y;
    }
    System.out.println("product is: " + seki);
  }
}
```

6　演習 9

疑似コードは前回の資料に載っていたので略。

```
import java.io.*;

public class r1ex9 {
  public static void main(String args[]) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("x = "); System.out.flush();
    int x = (new Integer(in.readLine())).intValue();
    System.out.print("y = "); System.out.flush();
    int y = (new Integer(in.readLine())).intValue();
    while(x != y) {
      if(x > y) {
        x = x - y;
```

```
      } else {
        y = y - x;
      }
    }
    System.out.println("GCD : " + x);
  }
}
```

なぜこれで最大公約数が求まるのだろう? 次のように考えてみるとよい。なお、x と y は正の整数であるも t のします。

- $x = y$ であれば、最大公約数は x そのもの。当然ですね。
- $x > y$ であれば、x と y の最大公約数は $x - y$ と y の最大公約数に等しい。[1]
- したがって、$x - y$ を改めて x とおいて、x と y の最大公約数を求めれば酔い。
- $x < y$ の場合も同様。
- この手順の反復ごとに、x または y のどちらかがより小さくなるが、0 以下にはならない (大きい方から小さい方を引くから)。
- ということは、この反復は有限回で止まる。
- ということは、そのとき $x = y$ が成り立ち、x が一番最初の x と y の最大公約数に等しい。

どうですか、繰り返しを使うときは「必ず止まって、止まった時には求める状況が成り立っている」ように設計する、という意味が分かります?

7 演習 9

これも疑似コードは略。

```
import java.io.*;

public class r1ex10 {
  public static void main(String args[]) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("x = "); System.out.flush();
    double x = (new Double(in.readLine())).doubleValue();
    double a = 0.0;
    double b = x;
    while((b - a) > 0.0000001) {
      double c = 0.5 * (a + b);
      if(c*c > x) {
        b = c;
      } else {
        a = c;
      }
    }
```

[1] 証明: 最大公約数を G とおくと、x も y も G の整数倍なのだから、$x - y$ もまた G の整数倍である。ということは、G は $x - y$ と y の公約数である (最大かどうかはまだ分からない)。ところで、もし最大公約数で「なかった」とすると、最大公約数 $H(> G)$ が別にあるわけで、H は y の約数かつ $x - y$ の約数。ということは、H は $x - y + y = x$ の約数でもある。これは G が x と y の最大公約数であるということに矛盾する。従って G は $x - y$ と y の最大公約数でもある。

```
      System.out.println("square root of " + x +" : " + a);
    }
  }
```

さて、このプログラムでなぜ「平方根を求める」ことができるか考えてみよう。ただし x は 1 より大きな数とする。

- x の平方根を R とすると、$0 \leq R < x$ である。
- $a = 0$、$b = x$ とおくのだから、$a \leq R < b$ である。
- $c = \frac{a+b}{2}$ としたとき、$c^2 > x$ であれば $a < R < c$ であるので、c を改めて b と置いたとき、$a \leq R < b$ である。そうでなければ、$c \leq R < b$ であるので、c を改めて a と置いたとき、$a \leq R < b$ である。
- $b - a$ は、反復ごとに a と b の平均 c を a、b のいずれかと置き換えるので、毎回半分ずつに小さくなっていく。ということは、いつかは必ず 0.0000001 以下になって停止する。
- このときも $a \leq R < b$ は成り立っているので、つまり a と R の差は 0.0000001、ということはその誤差以下で R が求まったと言える。

8 値とオブジェクト

前回もちょっと振れたが、重要な点なので再度値とオブジェクトの話をしよう。Java が扱うデータには大きく分けると「値」と「オブジェクト」の 2 種類があり、その区別は次のようになっている。

- 値 — 数値 (四則演算の対象になるもの) と、文字。四則演算程度の機能だけを持つ。単純なデータ。
- オブジェクト — さまざまな「もの」を表すデータ。込み入った構造や機能を持つことができる。クラスによって定義され (てい) る。言い替えればクラスとはオブジェクトの種類である。

値としては `int`(整数)、`long`(倍精度整数)`char`(文字)、`float`(実数)、`double`(倍精度 — 精度の高い — 実数)、などがある。一方、文字列などは複雑な構造を持つ (中に文字がたくさん詰まっている) のでオブジェクトで表す。ところが困ったことに、整数や文字などを表すオブジェクトも別途ある。これは「値」の方はデータの変換などの込み入った機能が入れられないため。そのような値とクラスを次に示しておく。クラス名は大文字で始まることに注意。

種別	値	クラス
真偽値	`boolean`	`Boolean`
バイト	`byte`	`Byte`
整数	`int`	`Integer`
倍精度整数	`long`	`Long`
文字	`char`	`Character`
実数	`float`	`Float`
倍精度実数	`double`	`Double`

基本的に、「値」に対してできることは各種の演算と自動的な文字列への変換だけで、それ以上の機能はすべて「クラス」の方の機能として用意されている。具体的な演算としては次のものがある。

四則演算	`+`	`-`	`*`	`/`	`%`	←剰余
ビット演算	`&`	`|`	`^`	`~`		
シフト演算	`<<`	`>>`				
論理演算	`&&`	`||`	`!`			
比較演算	`==`	`!=`	`<`	`>`	`<=`	`>=`
代入演算	`=`	`+=`	`-=`	`*=`	…	
増減演算	`++`	`--`				

ビット演算とシフト演算は整数 (計算機内部では 2 進数で、つまり 0 と 1 の並びで表されている) をビット列とみなして演算する。また、論理演算は「〜かつ〜」「〜または〜」「〜でない」を表す。たとえば次のように使う:

```
a > b && a > c    →  「a が b より大きく、かつ a が c より大きい」
x != 0 || !(a > c) →  「x が 0 でないか、または a が c より大きくない」
```

次に、クラスやオブジェクトはどうやって使うかについて説明しておこう。まず、クラスは「オブジェクトの種類」に対応していることを上で述べた。あるクラスに属する個々のオブジェクトをそのクラスの「インスタンス」と呼ぶ。インスタンスを作るには「new クラス名 (...)」という形の式を使う。

クラスのさまざまな機能を利用するには「メソッド」を呼ぶ。メソッドは次の 2 種類がある。

```
クラス名.メソッド名(...)    ←クラスメソッド
式.メソッド名(...)          ←インスタンスメソッド
```

「式」は何らかのインスタンスを計算するものでなければいけない。クラスメソッドとインスタンスメソッドの違いは、前者がクラスに所属していてインスタンスと関係を持たないのに対し、後者はインスタンスに所属している (たとえば「私」オブジェクトの「名前を返す」メソッドの結果は「あなた」オブジェクトの「名前を返す」メソッドの結果とは違う) ことである。

WWW 上で「JDK 1.1 API」のドキュメントを参照すると、標準のクラス群としてどのようなものがあるかを見ることができる (これを API ドキュメントと呼ぶ)。ドキュメントは「パッケージ」と呼ばれる単位ごとに分かれている。たとえば PrintStream クラス (System.out には PrintStream のインスタンスが入っている) は java.io パッケージに入っている。すべてのクラス名は曖昧さがないようにパッケージ名をつけた「java.io.PrintStream」のような形で指定してもよい。

演習 1 API ドキュメント中の java.io.PrintStream の箇所を眺めてみよ。知っている/使ったことのあるメソッドの説明を読め。

演習 2 API ドキュメント中の java.lang.String の箇所を眺めてみよ。文字列に対してだいたいどのような操作ができるのか自分なりに整理してみよ。

9 String オブジェクト

さて、お話ばかりではつまらないので、とりあえずよくお目に掛かって色々役に立つクラスである String クラスを題材にちょっと遊んでみよう。String は文字列を表すクラスである (これに対し、文字は char 型の値。「"..."」は文字列。「'...'」は文字)。

まずは「左右ひっくり返し」の例題を見てみよう。疑似コードを示す。

- 無限に繰り返し:
- 　文字列 str を入力。
- 　str が空文字列なら、ループを抜け出す。
- 　res ← 空文字列。
- 　i を str の長さ-1 から 0 まで変えながら繰り返し:
- 　　res の末尾に str の i 番目の文字を連結。
- 　ここまで繰り返し。
- 　res を出力。
- ここまで繰り返し。

先にも述べたように、計算機では数は 0 から数えることが多いので、文字列も長さ L の文字列であれば、その中には 0 番目、1 番目、…、$L-1$ 番目の文字が含まれると考える。Java のコードは次の通り。

```
import java.io.*;

public class R2Sample1 {
  public static void main(String args[]) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    while(true) {
      System.out.print("String> "); System.out.flush();
      String str = in.readLine();
      if(str.equals("")) break;
      String res = "";
      for(int i = str.length()-1; i >= 0; --i) {
        res += str.charAt(i);
      }
      System.out.println(res);
    }
  }
}
```

では説明を。

- 1 回入力したらおしまいではつまらないので、このプログラムは空行 ([RET] のみ) を入力するまで繰り返し処理を行う。
- 「while(true) ... 」では、条件が「真」という定数なので無限に繰り返すことになる。
- 文字列はオブジェクトなので「==」では比較できない。メソッド `equals()` を使うこと。ここでは空文字列と比較している。
- `break` というのは新しい文で「ループから抜け出す」という動作を行う。無限ループなのでこれがないと終わらない。なお、if 文の枝として文が 1 つしかない場合は「{ ... } 」で囲まないでもよい。
- 文字列の長さはメソッド `length()` で調べられる。ここでは for 文を「$L-1$ から 0 まで順に繰り返し」で使うので i を 1 つずつ減らして行く。
- くっつけるのは「+」だが、ここでは「くっつけ込む」ので「+=」を使っている。

動かしてみよう。

```
% javac R2Sample1.java
% java R2Sample1
String> baka
akab
String> aho
oha
String>    ← [RET] だけを打った
%
```

なかなか面白いでしょう?

演習 3 上の例題をそのまま打ち込んで動かせ。

演習 4 次の Java プログラムを作れ。String クラスのメソッドを活用すること。

a. 入力文字列をすべて大文字にして表示する。(ヒント: toUpperCase() を使う)
b. 入力文字列に現われる「a」をすべて「*」にする。(ヒント: replace() を使う)
c. 入力文字列に現われる「a、e、i、o、u」をすべて「*」にする。(ヒント: replace() を 5 回使う)

d〜f. 次のような三角形や巡回を表示する。(ヒント: substring() を使う)

```
d: String> abcd  e: String> abcd  f: String>abcd
   abcd             abcd             bcda
   abc               bcd             cdab
   ab                 cd             dabc
   a                   d             abcd
```

10 配列

本日最後の話題として、配列について説明しよう。先に出て来た文字列は「文字の並び」を表す特別なクラスだったが、「整数の並んだもの」や「実数の並んだもの」等、一般に「同じ型が並んだもの」を表すプログラミング言語の機能を「配列」と呼ぶ。

Javaでは配列の型は元の型の後ろに「`[]`」をつけて表す。たとえば「`int[]`」は整数の並んだ配列、「`String[]`」は文字列の並んだ配列ということになる。

次に、配列を使うにはその配列型の変数を宣言した上で、大きさを指定して配列オブジェクトを用意しなければならない。具体的には次のような感じになる。

```
int[] a = new int[100];  ←要素数 100 の配列を用意
```

「new」を使うことから分かるように、配列も一種のオブジェクトである (ただし書き方がやや特別な形になっている)。

いちど用意してしまえば、配列の個々の要素は 1 つの変数と同様に扱える。ここで「どの要素か」を指定するのに `[...]` の中に式を書いて指定する (これを添字と呼ぶ)。たとえば上の例だと `a[0]`〜`a[99]` という要素があることになる (例によって 0 番目から数えるのに注意)。

では、配列に整数をいくつか (0 が現われるまで、ただし最大 100 個まで) 読み込み、それを逆順に表示するという例題。

- a ← 大きさ 100 の整数配列。
- count ← 0。
- count+1 < a の要素数である間繰り返し:
- 　v に整数を入力する。
- 　もし v = 0 ならば、繰り返しを終わる。
- 　a[count] ← v。
- 　count を 1 ふやす。
- ここまで繰り返し。
- 変数 i を count-1 から 0 まで変えながら繰り返し:
- 　a[i] を出力する。
- ここまで繰り返し。

なお、最初のループの条件は「次に 1 個増やしたらもう配列に入り切らなくなるならループをやめる」ということ。このように、配列は最初に大きさを指定して作ってしまうので、指定した以上の要素を使わないように注意しながらプログラムを作る必要がある。Java コードは次の通り。

```
import java.io.*;

public class R2Sample2 {
  public static void main(String args[]) throws Exception {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    int[] a = new int[100];
    int count = 0;
```

```
    while(count+1 < a.length) {
      System.out.print("" + count + "> "); System.out.flush();
      int v = (new Integer(in.readLine())).intValue();
      if(v == 0) break;
      a[count] = v;
      ++count;
    }
    for(int i = count-1; i >= 0; --i) {
      System.out.println(a[i]);
    }
  }
}
```

なお、「a.length」というのは、a が配列オブジェクトの場合、その変数 length を調べれば配列の要素数が分かるのでこれを参照しているもの。

演習 5 上記のプログラムをそのまま打ち込んで動かせ。

演習 6 下記のような Java プログラムを作れ。

a. 配列に整数を読み込み、その最大値と最少値を出力する。

b. 配列に整数を読み込み、その合計と平均を出力する。

c. 配列に整数を読み込み、その平均より大きい要素を出力する。

演習 7 下記の疑似コードは、配列に整数を読み込み、それを小さい順に並べて出力するものである。この疑似コードを Java に直して動かせ。またなぜこれで小さい順に並べて出力できるのか説明せよ。

- a ← 大きさ 100 の整数配列。
- count ← 0。
- count+1 < a の要素数である間繰り返し:
- 　v に整数を入力する。
- 　もし v = 0 ならば、繰り返しを終わる。
- 　a[count] ← v。
- 　count を 1 ふやす。
- ここまで繰り返し。
- 変数 i を 0 から count-1 まで変えながら繰り返し:
- 　変数 k を 1 から count-1 まで変えながら繰り返し:
- 　　もし a[k-1] > a[k] ならば、
- 　　x ← a[k-1]。
- 　　a[k-1] ← a[k]。
- 　　a[k] ← x。
- 　枝分かれ終わり。
- 　ここまで繰り返し。
- ここまで繰り返し。
- 変数 i を 0 から count-1 まで変えながら繰り返し:
- 　a[i] を出力。
- ここまで繰り返し。