（2008-05-20 版）

Java によるプログラミング入門 (4)

1　Lesson 2: 制御構造と配列

問題意識　時間 time の値を何度も変えて水位 tankLevel を計算することを考えます．同じような動作を繰り返したり，状況に応じて動作を変えたりするために本章では「制御構造」を学びます．また，大量のデータを扱うために「配列」も学びます．

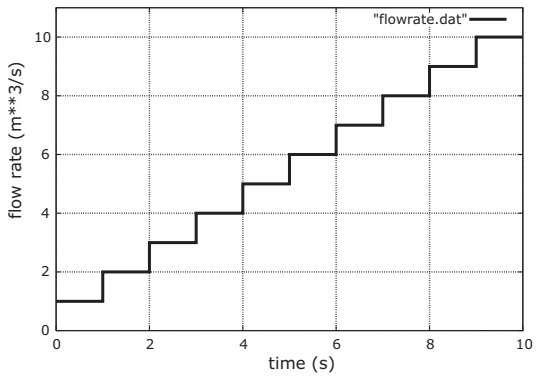


図 1　時間変化する流入量

1.1　タンクへの流入量が時間とともに変わる場合の水位を計算する

問題は次のようなものです：

流入量が最初の 1 秒間は $1m^3/s$, 次の 1 秒間は $2m^3/s$, と毎秒 $1m^3/s$ づつ 10 秒間にわたって階段状に増加する場合の各時刻の水位を求めよ．[1]

解き方の方針としては次の式を時間範囲で繰り返して計算します：

$$1\text{ 秒後の水位} = \text{現在の水位} + \text{現時刻の流入量}/\text{タンクの底面積}\ l_{t+1} = l_t + f_t/a \quad (1)$$

単純には，TankCalculator.java の記述を長くして，必要な時間の分だけ同じような命令を繰り返せば実現できますがあまり効率的な手法とは思えません．そこで

- 時間の添え字のついた変数（f_t みたいなもの）を**配列**として用意する．
- 同じ数式を繰り返すためにプログラムの**流れを制御する命令**を使う．

[1] ここではシミュレーションを容易にするために階段状の変化を考えます．

という方法で実現します.

1.2 プログラム TankCalculator2.java

```
 1: public class TankCalculator2 {
 2:   public static void main(String args[]) {
 3:     final double TANK_AREA = 20.0;
 4:     final double INITIAL_LEVEL = 10.0;
 5:     double flow[]
 6:       = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 0.0};
 7:     double tankLevel[] = new double[flow.length];
 8:
 9:     tankLevel[0] = INITIAL_LEVEL;
10:     for (int i = 0; i< flow.length-1; i++) {
11:       tankLevel[i+1] = tankLevel[i] + flow[i]/TANK_AREA;
12:     }
13:     System.out.println("Time(s), Flow(m**3/s), Tank Level(m)");
14:     for (int i = 0; i< flow.length; i++) {
15:       double time = i*1.0;
16:       System.out.println(time + ", " + flow[i] + ", " + tankLevel[i]);
17:     }
18:   }
19: }
```

上のプログラムは入力してもらった TankCalculator2.java です．これを含め，本日使うファイルを http://130.54.13.233/lecture/java/lesson234.zip からダウンロードし，解凍，M ドライブの java フォルダに入れてください．

プログラムの要点を以下に説明します．

- 5,6 行目：時間とともに変化する流入量を double 型の配列（ハコが並んだもの） flow に設定します．
- 7 行目：時間とともに変化する水位を格納する double 型の配列 tankLevel を確保します．flow.length は配列の長さ（この場合 11）です．
- 9 行目：水位を初期化します．
- 10〜12 行目：i=0 から i ＜ 10 の間，1 ずつ i が増えて 10 回の繰り返し計算を行い，水位を計算します．
- 11 行目：水位を更新する漸化式です．
- 13〜17 行目：時刻・流量・水位を出力します．

ここから先は，ここに登場した「配列」「繰り返し計算」についてもう少しフォーマルに説明していきます．内容が多いので，補足説明の部分は「そんな考え方もあるんだな」程度に理解していただければ結構です．

1.3 プログラムの流れを制御する — 条件式

プログラムは，基本的には上から下に一行ずつ実行されますが，変数などの値によってプログラムの流れを制御することもできます．

そのためには，特定の条件が満たされたかどうかを判断する**条件式**を用います．条件式は関係演算子や論理演算子を使って記述される式でその値は boolean 型 (とる値は true と false) になります．

数値の比較 ：数値を比較する演算子は`==`，`<`，`>`，`<=`，`>=`，`!=` などがあります[2]．

論理演算 ：AND や OR などの論理演算子は `&&`，`||`，`!`
(それぞれ AND, OR, NOT) などがあります．

例えば変数 i の値が 10 以上，20 未満であることを判定するための式は

```
(10 <= i) && (i < 20)
```

のように書きます．

足し算よりかけ算を優先するように，演算子には優先順位がつけられていますが，間違いを防ぐには () を使って意味を明確にすべきです．

[2] 等しいかどうかの判定は == です．代入の= と間違わないようにしましょう．

1.4　プログラムの流れを制御する— if 文

条件によって動作を変えるには if 文を用います．以下の 3 通りの使い方を押さえておきましょう．

1. 条件が成立したときのみ処理を実行

```
if（条件）{
  処理
}
```

2. 条件が成立したときに処理 1 を成立しないときに処理 2 を実行

```
if（条件）{
  処理 1
} else {
  処理 2
}
```

3. 条件 1 が成立しないときにさらに条件 2 にしたがって処理を変える

```
if（条件 1）{
  処理 1
} else if（条件 2){
  処理 2
} else {
  処理 3
}
```

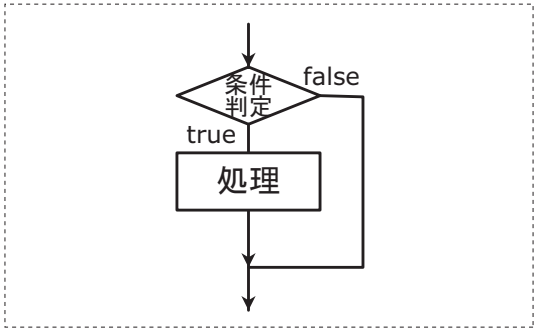


図 2　if 文の実行の流れ 1

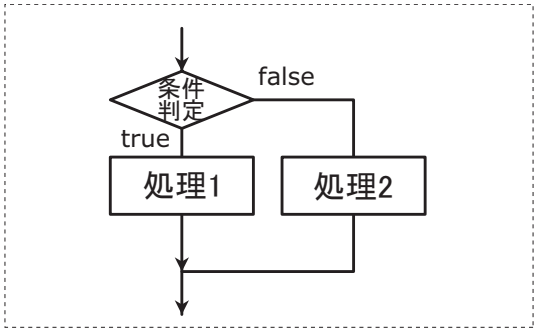


図 3　if 文の実行の流れ 2

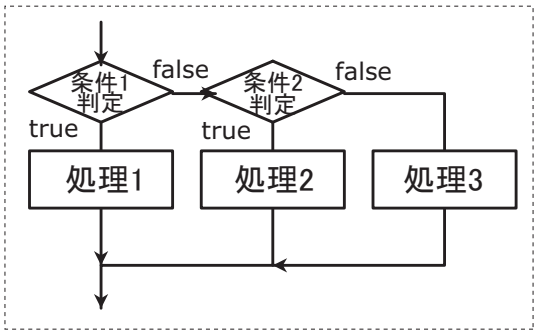


図 4　if 文の実行の流れ 3

1.5　プログラムの流れを制御する— for 文

プログラムでしばしば現れるのは同様の動作を繰り返すことです．このような繰返しのための命令（制御構造）が for 文です．for 文は以下の様に記述します．

```
for（前処理；ループ内実行前の終了判定；ループ内実行後の後処理）{
    ループ内の実行命令
}
```

たとえば変数 i を 0 から 9 まで変えて繰り返すときは次のように書きます．for 文ではいろいろな処理を書くことができますが，このような使い方が最も多いです．これは定型として覚えましょう．

```
for (int i = 0; i < 10; i++) {
    // ここに繰返し実行することを書く.
}
```

- 後処理の“ i++ ”の“++”はインクリメント演算子と呼ばれ，変数 i の値を 1 増やします．i = i + 1 あるいは i += 1 と同じです[3]．
- 終了条件を i < 10 と書くか i <= 9 は場合によりますが，統一しておくのが望ましいです．

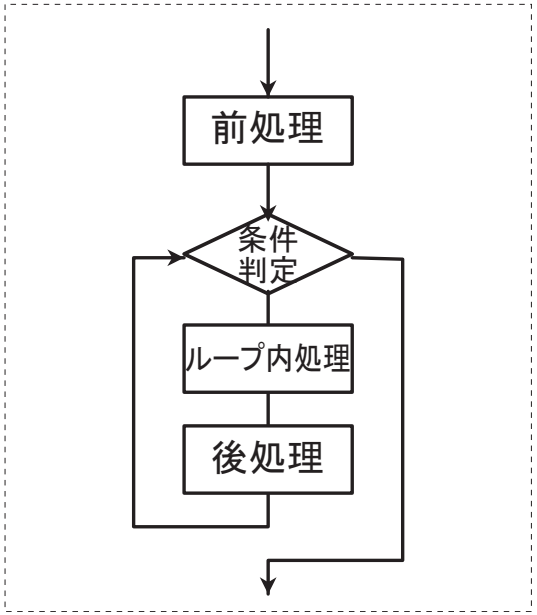


図 5　for 文の実行の流れ

[3] 値を 1 減らす“--”もあります．

1.6 プログラムの流れを制御する— while 文

特に前処理や後処理が必要ない場合の制御構造が while 文です.

```
while（ループ内実行前の終了判定）{
    ループ内の実行命令
}
```

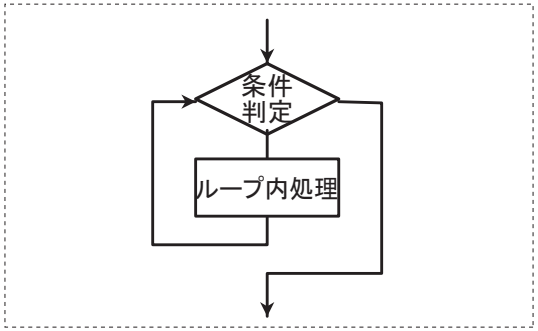


図 6　while 文の実行の流れ

条件判定はループ内の実行前に行われます．最初に判定で条件式が false ならばループ内は一度も実行されません.

【ループ脱出】ループは “break” 命令を用いて途中で抜けられます.

```
for (int i = 0; i < 10; i++) {
    ...     // ここまでは実行される
    break; // ここでループから脱出.
}
```

【例】ピンと来る (?) 例を挙げておきます.

```
while（所持金 > 0）{
    馬券を買う;
    if（嫁さんから電話があった）{
        break;
    }
}
```

【for 文と while 文の関係】先の for 文の例は while 文では以下のようにかけます.

```
int i = 0;
while(i<10) {
  ループ内の実行命令
  i++;
}
```

【演習】1 から 100 までの，7 の倍数を表示するプログラムを書いてください.

- SevenFor.java をテンプレとしてお使いください.
- 「7 で割った余りが 0 であること」は，**変数** `% 7 == 0` と書きます.
- いろいろな方法があります．余裕のある人は while 文なども使いいろいろ書いてみてください.

【演習】2 から 100 までの，素数を表示するプログラム Sosuu.java を完成させてください.（自信があれば，テンプレを見ずに作ってみてください）

1.6.1 補足１．ブロック

for, while, if 命令で使われる “{” と “}” で囲まれた部分を**ブロック**といい，その中の命令はひとまとめに扱われます．ブロック内の命令は，通常字下げします．

```
// ブロックを使う例
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

【発展】ブロック内の命令が 1 つだけなら，ブロックを使わずに書くこともできます：

```
for (int i = 0; i < 10; i++)
    System.out.println(i);
```

しかしこのような書き方は後で for ループ内の処理を追加しようとしたときに

```
for (int i = 0; i < 10; i++)
    System.out.println(i);
    追加した命令; // ここは for ループの外側
```

のように誤る懸念があるため**推奨されません**[4]．以下のように書くべきです．

```
for (int i = 0; i < 10; i++) {
    System.out.println(i);
    追加した命令; // ここは for ループの外側
}
```

[4] 字下げをしてあると一見，正しく見えてしまいますが，Java では字下げは見やすくするだけで実行上の効果はありません．

1.6.2　補足２．変数のスコープ

問題意識 日常の生活でもメモ用紙などに情報を記録しますが，それらのメモを捨てずに置いておくとどのようなことが生じるでしょうか．情報がどんどん混乱して別の作業を行う際に支障をきたすでしょう．仕事が終わったら，不要になったメモを捨てなくてはなりません．

プログラムでは，一時的に利用する変数を「その場かぎりのもの」として確保し，作業が終わると捨て去るための「変数のスコープ」という仕組みがあります．たとえば

```
for (int i = 0; i < 10; i++) {
  // ループ内の処理
}
```

の中で，ループのカウンタとして使う変数 i は for 文内で宣言されているので for 文の中でのみ参照可能で，for 文が終了すると捨て去るように作られています．

基本的なルールは次のようなものです：

- 変数は宣言されたときから利用可能となり，
- その宣言が含まれているブロックが終了したときに捨て去られる．
- 逆に，ブロックの中で得た計算結果をブロックの外側で利用するためには，その変数はあらかじめブロックの外側で宣言されている必要がある．

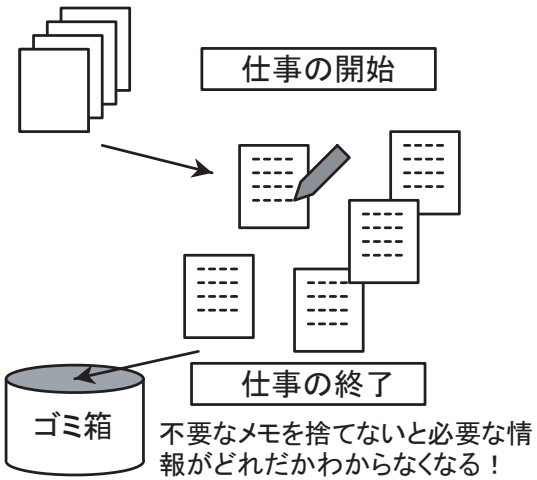


図 7　作業メモの扱い

1.6.3 補足３．読みやすいコードを書くための工夫

for, while, if などの制御構造を含むプログラムを読みやすくするためには以下のような点に注意しましょう．良いプログラムを写経することで身に付いていきます．

- 条件文は () などを使って分かりやすく書く．
- for, while, if などではブロックを利用する．
- 1 行だけのブロックでも {} で囲む．
- ブロックの内側はブロック内であることを分かりやすくするため字下げする．
- カウンタの値を変化させる for 文の記述では
  - 記述ルール（終了条件の不等号など）を統一する．
  - スコープを意識して変数を利用する．
  - ループのカウンタに使う変数をできるだけ固定する (i や j が好まれて使われます)．例えば次の例のように，入れ子になったループは i, j の順に使うなどです．

```
for (int i = 0; i < 10; i++) {
  for (int j = 0; j < 10; j++) {
    ループ内の処理;
  }
}
```

1.7　配列 — データを一括して扱う

多くのデータを効果的に扱う仕組みが配列です[5].

- **【配列の宣言と確保１】**ある型の変数に [] をつけることで，その型の配列として宣言できます．初期値を与えて配列を作るには次のように書きます．

```
double flow[] = {1.0, 2.0, 3.0, 4.0, 5.0};
```

この場合，flow[0] が 1.0 ，flow[1] が 2.0 になります．

- **【配列の宣言と確保２】**値は与えずに一定の要素数の配列を作るには

```
double tankLevel[] = new double[11];
```

と書きます，配列の宣言と確保を分けて次のように書くこともできます．

```
double tankLevel[];
tankLevel = new double[11];
```

- **【配列の長さ】** 配列の長さは配列変数名に “.length” を付けて得ます．flow と要素数の多い配列を作るためには次のように書きます：

```
double tankLevel[] = new double[flow.length];
```

- **【範囲】**配列の添え字の範囲は **0 から “長さ–1” まで**です．for 文で配列の内容を操作する時の書き方は定型として覚えておきましょう．

```
for (int i = 0; i < flow.length; i++) {
  System.out.println(flow[i]);
}
```

[5] 配列は横にならんだ複数のハコのようなものだと考えてください．ハコには，0 から始まる番号が付いています．

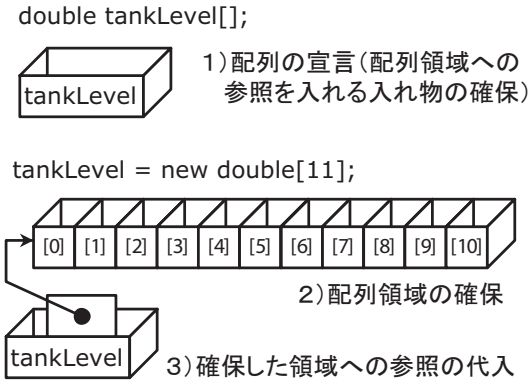


図 8　配列の宣言と確保

1.8　配列を扱うときの注意

問題意識 文書などの情報を人に伝える際にどのようにしているでしょうか．

一つのやり方は文書そのものを**コピーして渡す**ことです．この場合，コピーをもらったBさんはそのコピーを自由に扱え，どんなに手を加えても原本に影響は与えません．しかしもし文書が 100 ページもあったらコピーは大変です．

もう一つのやり方は**文書の所在（参照）を伝える**ことです．必要なときに見に来てもらう，という考え方です．この方法だと，コピーの手間が不要な一方，Bさんの作業がAさんにも影響を与えてしまうという危険が伴います．

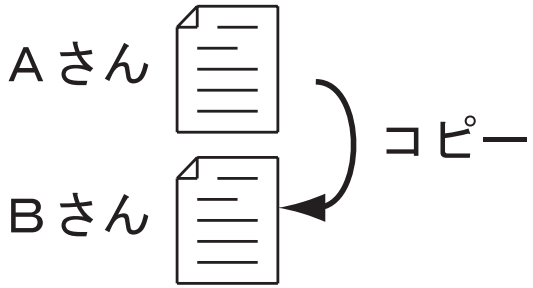


図 9　コピーして渡す

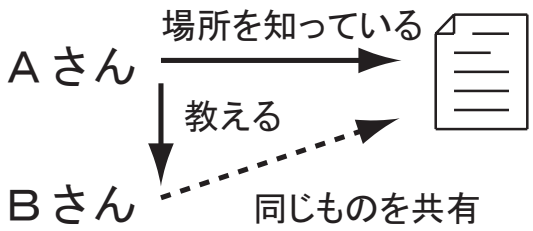


図 10　場所を伝える

java での変数はその型によって上に述べたように値そのものを扱う場合と**参照**を扱う場合の相違があります．

- **int や double は値**：int 型や double 型の変数は変数の値そのものと一体です．これはコピーして渡されます．
- **配列型は参照**：配列型では，データそのものは別の場所にあり，変数はその参照だけを持ちます．

```
double array[];          //「参照」を持つ場所を作っている
array = new double[3];   //データそのものを作り，その参照を設定
                         している．
```

配列変数への代入は要注意：

次の例のように「配列の代入」は「配列の内容の代入」ではなく，**「配列への参照の代入」**になります．先ほどの例で言えば,「ＡさんがＢさんにコピーを渡す」のではなく,「ＡさんがＢさんに情報の所在を教えた」状態です．

```
int array1[] = {1, 2, 3};
int array2[] = {4, 5, 6};
// ここまでは array1 と array2 は別のものを指している．
array2 = array1;
// 配列の代入により array2 は array1 と同じものを指す．
// {1, 2, 3} がコピーされるわけではない．
array2[0] = 7;
//array2[0] に 7 を代入すると array1[0] の値も 7 になる．
//array2 に対する処理が，array1 にも影響を与えてしまう．
```

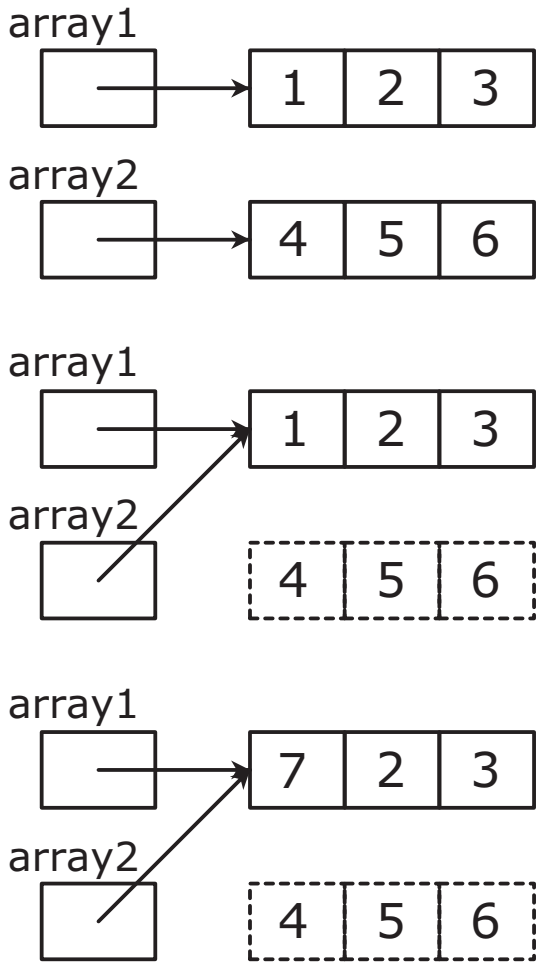


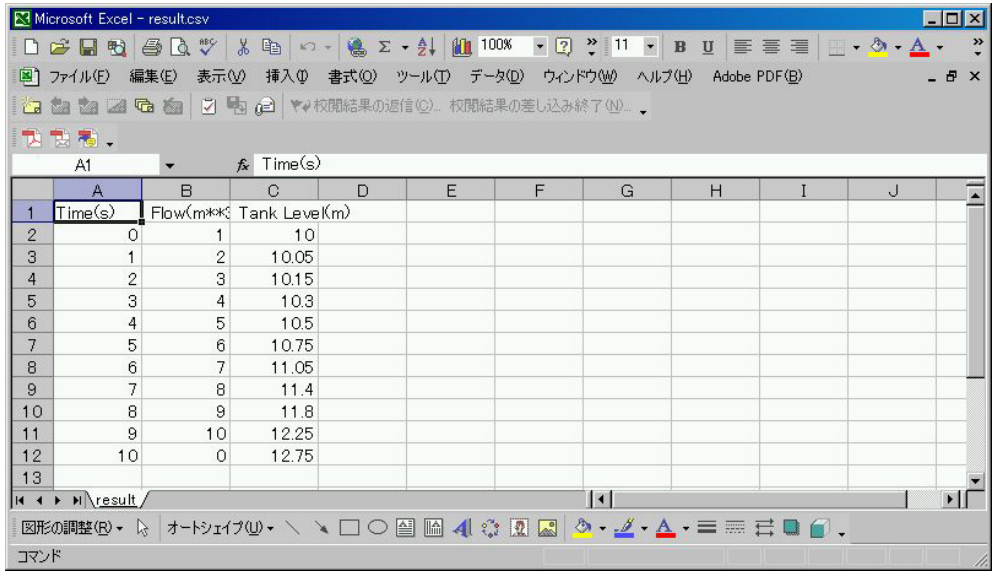
図 11　配列の代入

1.8.1　補足４．CSV フォーマット（演習）

数値や文字列を “,” (カンマ) で区切ったテキストファイルの形式を CSV フォーマットと呼びます．この形式はプログラムで比較的容易に生成でき，Excel などの表計算ソフトで読み込んでさまざまな処理ができます．Windows では拡張子 “csv” を付けておくことで，ダブルクリックにより Excel で開くことができます．

```
java TankCalculator2 > result.csv
```

最初の行に各欄の名称などを，2 行目以降にデータの値を書くようにすると表計算ソフトでの処理がやりやすいようです．実際にやってみてください．



	A	B	C
1	Time(s)	Flow(m*k	Tank Level(m)
2	0	1	10
3	1	2	10.05
4	2	3	10.15
5	3	4	10.3
6	4	5	10.5
7	5	6	10.75
8	6	7	11.05
9	7	8	11.4
10	8	9	11.8
11	9	10	12.25
12	10	0	12.75
13			

図 12　表計算ソフトでの読み込み

1.8.2　補足５．２次元配列

数学の行列や表計算ソフトのシートのように 2 次元の配列も扱えます．また 2 次元配列の要素を操作するには for 文を入れ子にして使うことが多いです．例えば以下の例では行列を 2 次元配列で表して単位行列（対角線上の成分だけが 1 で他は 0 の行列を作るものです．

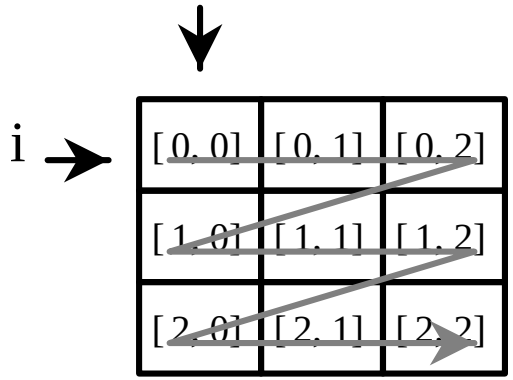


図 13　2 重 for ループによる行列（2 次元配列）の走査

```
// 大きさを 3 に．
int size = 3;
// 2 次元配列を確保
double a[][] = new double[size][size];
// 要素をすべて 0 に
for (int i = 0; i < size; i++) {
    for(int j = 0; j < size; j++) {
        a[i][j] = 0.0;
    }
}
// 対角要素を 1 に
for (int i = 0; i<size; i++) {
    a[i][i] = 1.0;
}
```