計算機プログラミング I (6)

メソッドによる処理の抽象化

2000 年 11 月 14 日 増原 英彦

プログラミング言語には、メソッド(method)・手続き(procedure)・関数(function)・サブルーチン(subroutine)などと呼ばれる[1]ものがあり、ひとかたまりの処理に名前をつけて使うことができる。Java 言語には、メソッドを定義して使う機能がある。メソッドを使うこで、次のようなことが可能になる:

- 大きな問題を解決する方法の基本は、小さな問題に分割してそれぞれを解くことである。大きなプログラムを作る場合も同様に、様々な処理を分割して書くことが重要である。メソッドを使うことで処理を分割して書くことができる。
- 色々なプログラムに共通する処理があった場合、それらを毎回書かずに融通できるようにしたい。メソッドを使えば、他のプログラムのために作られた処理を利用することができる。

まずは、メソッドを使わない場合にどのような問題があるかを見てみよう。

1 例題: 対角線長の標準偏差

例として、「n 個の長方形の幅 (w_i) と高さ (h_i) がそれぞれ与えられたとき、それらの対角線の長さの標準偏差を求める」という問題を考える。i 番目の長方形の対角線長は、三平方の定理より

$$d_i = \sqrt{w_i^2 + h_i^2} \tag{1}$$

である。また、$d_i (i = 1, \ldots, n)$ の標準偏差は

$$\sigma = \sqrt{\overline{(d_i^2)} - \left(\overline{d_i}\right)^2} = \sqrt{\frac{\sum d_i^2}{n} - \left(\frac{\sum d_i}{n}\right)^2} \tag{2}$$

(ただし、$\overline{x_i}$ は x_i の平均値) である。プログラムとしては、「対角線長の 2 乗」の合計と「対角線長」の合計を繰り返しによって求め、式 (2) の右辺に従って計算をするので、図 1 のようになる。ここで平方根は、［*** ***］で囲んだ部分は、課題 1–2 で行ったような漸近解法による平方根の計算を書くことにする[2]。

実際に［*** ***］の部分に平方根を求める処理を書いたプログラムは、式 (2) よりかなり複雑なものになる ($\Rightarrow$ web ページのプログラム)。その原因として、2 つの要素がある:

- 式 (2) では、$\sqrt{}$ という記号 1 つで済んでいた平方根の計算が、プログラム中で大きな場所を占めている。つまり、標準偏差のアルゴリズムにとって本質的には重要でないものにも関らず、プログラムに占める割合が大きくなっている。
- 式 (1) と式 (2) の平方根の計算が、プログラム中で別々に書かれている。つまり、同じアルゴリズムが 2ヶ所に書かれている。

結果として、プログラムを人間が読んで理解する、誤りを修正する、より良いアルゴリズムに改良する、といったことが難しくなってしまっていることが分かるだろう。

[1] 言語によって異なる名前が付いているが、どれも同じようなものである。

[2] 「ライブラリ関数 `Math.sqrt(x)` を使えばいいじゃん」と思う人もいるかも知れない。この例に限ってはその通りである。しかし、`Math.sqrt(x)` は誰かが定義しているのであって、その方法を学ぶのが今週の目標なのである。

```
1  // 長方形の対角線長の標準偏差を計算する
2  double widths[] = new double[] { 20, 18, 24, 31, 17 }; // 長方形の幅
3  double heights[] = new double[] { 4,  5,  7,  2,  3 }; // 長方形の高さ
4  int size = widths.length; // 長方形の個数
5
6  double diagonalSum = 0;      // 対角線長の合計を覚える変数
7  double diagonalSqrSum = 0;  // 対角線長の 2 乗の合計を覚える変数
8  // 各長方形の対角線長と 2 乗値を計算しつつ合計を求める
9  for (int i = 0; i < size; i=i+1) {
10   // 三平方定理より、対角線長の 2 乗を計算
11   double diagonalSqr = widths[i] の*widths[i] + heights[i]*heights[i];
12   diagonalSqrSum = diagonalSqrSum + diagonalSqr; // 対角線長の 2 乗の合計に加算
13   double diagonal;                               // 対角線長を覚える変数
14   [*** diagonalSqr の平方根を計算し、diagonal に代入 ***]
15   diagonalSum = diagonalSum + diagonal;          // 対角線長の合計に加算
16 }
17
18 double diagonalAve = diagonalSum/size; // 対角線長の平均値
19 double stdDevSqr = diagonalSqrSum/size - diagonalAve*diagonalAve;// 標準偏差の 2 乗値
20 double stdDev;                          // 対角線長の標準偏差覚える変数
21 [*** stdDevSqr の平方根を計算し、stdDev に代入 ***]
22 System.out.println("対角線長の標準偏差は"+stdDev+"です。");
```

図 1: 対角線長の標準偏差を求めるプログラム (完全版は web ページにある)

2 手続きの抽象化

同じアルゴリズムを何箇所にも書くかわりに、一つの定義で済ますためにはメソッドを使う。まずは、「横棒を表示する」といった処理の手順をメソッドとして定義して使う方法を見てみよう。

例えば画面にメッセージの前後を横棒ではさんで表示するプログラムを作ることを考えよう。横棒を表示するためには繰り返しを使って同じ文字を何個も書けばよいのだが、メッセージの前後で同じことをするためにメソッドを使う。

```
1  class Messenger { // メッセージの表示
2    static void horizontalBar() { // 横棒を表示するメソッド
3      int length = 25;            // 棒の長さ
4      for (int count = 0; count < length; count = count + 1)
5        System.out.print("=");    // 「=」を改行せずに表示
6      System.out.println();       // 最後に改行
7    }
8    public static void main(String[] args) { // プログラムの入口
9      Messenger.horizontalBar();                  // 横棒を表示
10     System.out.println("重要なお知らせです。"); // メッセージの
11     System.out.println();                       // 表示
12     System.out.println("情報教育棟内では飲食は");
13     System.out.println("禁止されています。");
14     Messenger.horizontalBar();                  // 横棒を表示
15   }
16 }
```

メソッドの定義: 2–7 行目が horizontalBar という名前のメソッドを定義している部分である。

- static は、メソッドが「クラスに定義されたもの」であることを示している。しばらくの間は必ず static と書くものと思っていてよい。
- void は、メソッドが答えを返す値の型を示している。表示などの処理をするだけで答えを返さない場合は void と書く。(答えを返す場合は 4 節で説明する。)

- horizontalBar はメソッドの名前である。メソッドの名前は変数名と同様、アルファベットと数字の並びで、分かりやすい名前をつけるべきである。
- () の間にはメソッドが情報を受け取るための引数 (ひきすう) を書く。(3 節で説明する。)
- 引数に続いて{ }で囲まれた部分つまり、3–6 行目が、メソッドの本文である。

使用: 9,14 行目ではメソッドを実行している。メソッドを使う式は、Messenger.horizontalBar() のように、クラス名.メソッド名(引数, 引数, ...) のように書く。(ここでは、この式が文になるので最後に「;」がついている。)

働き: メソッドを使う式は、そこにそのメソッドの本文が書かれていたかのようになる。このプログラム実行は次のようになる。

- まず、main メソッドの先頭の文から実行が始まる。
- メソッドを使う式 (Messenger.horizontalBar()) があった場合、その式のかわりにそのメソッドの本文が書かれていたかのように実行を続ける。
- メソッドの本文の実行が終ったら、もとのメソッド (main) の実行を再開する。

3 メソッドの動きを変える

3.1 引数

上の例で、メソッドを使う側 (main) で、表示される横棒の長さや横棒に使われる文字を変えるにはどうすればよいのだろうか? メソッドの引数(ひきすう) を使えば、メソッドの使用側からメソッドの定義側に「長さ」や「使う文字」といった情報を引数として渡すことができる。

```
1  class PMessenger { // 引数付メソッド
2    // length 個の brick から成る横棒を表示して改行
3    static void horizontalBar(int length, String brick) {
4      for (int count = 0; count < length; count = count + 1)
5        System.out.print(brick);
6      System.out.println(); // 最後に改行
7    }
8
9    public static void main(String[] args) {
10     PMessenger.horizontalBar(11,"▼"); // 横棒を表示
11     System.out.println(" 重要なお知らせです。");
12     System.out.println();
13     System.out.println("情報教育棟内では飲食は");
14     System.out.println("    禁止されています。");
15     PMessenger.horizontalBar(11,"▲"); // 横棒を表示
16   }
17 }
```

メソッドの定義: メソッド定義では、horizontalBar(int length, String brick) のように( ) 内に、情報を受け取るための変数名とその型を書く。ここでは、「横棒の長さ」を整数型 (int) の変数 length で、「使う文字」を文字列型 (String) の変数 brick で受け取ることを宣言している。このような変数を仮引数ということもある。

メソッドの使用: メソッドを使用する側では、PMessenger.horizontalBar(11,"▼") のように、渡したい情報を( ) 内に書く。ここでは、長さとして「11」、使う文字列として「"▼"」を渡している。このような式を引数(あるいは実引数) と呼んだりパラメータと呼ぶことがある。

働き: 引数のあるメソッドを使う式はおおまかに言って次のように実行される:

1. 引数( ) 内の式 (「11」と「"▼ "」) が計算される
2. メソッド定義側の仮引数 (length と brick) に、計算された値が入れられる
3. メソッド本体が実行される

例えば 10 行目のメソッドの使用PMessenger.horizontalBar(11,"▼ "); のあたりは、下のようなプログラムとほぼ同になる:

```
1  public static void main(String[] args) {
2    // PMessenger.horizontalBar の中身 (ここから)
3    int length=11;       // 仮引数に値がセットされる
4    String brick="▼ ";  //          〃
5    for (int count = 0; count < length; count = count + 1)
6      System.out.print(brick);
7    System.out.println(); // 最後に改行
8    // PMessenger.horizontalBar の中身 (ここまで)
9    System.out.println(" 重要なお知らせです。"); // main の続き
10   ...(以下略)...
```

3.2 変数の有効範囲

メソッドの中で定義される変数は、定義している側と使う側で全く別のものと見なされる。例えば下のようなプログラムを考えてみる。

```
1  // メソッドの定義側
2  static void horizontalBar(int length, String brick) {
3    length = length + 1; // 仮引数 length を変更する
4    ... (略) ...
5  }
6
7  public static void main(String[] args) {
8    int length=11;                          // 同じ名前の変数を定義して
9    PMessenger.horizontalBar(length,"▼ "); // 使う
10   ... (略) ...
11   PMessenger.horizontalBar(length,"▲ "); // もう一度使う
12 }
```

このプログラムでは、定義側 (horizontalBar) と使用側 (main) で字面上は同じ length という名前の変数を使っているが、実行においては別の変数として扱われる。そのため、定義側で length=length+1; が実行されても使用側の length に入っている値は 11 のまま変わらない。

この性質は、小さなメソッドを組み合わせて大きなプログラムを作る場合に、複数のメソッドが偶然同じ名前の変数を使っても問題なく動くために重要である。

練習 **6-1:** (変数の有効範囲)

```
1  static void bar(int length, String brick) { // 長さ length の横棒を描く
2    int i = 0 ;
3    for (i = 0; i < length; i=i+1)
4      System.out.print(brick);
5    System.out.println();
6  }
7  public static void main(String[] args) { // 三角形を描く
8    int length = 5;
9    int i = 0;
10   for (i = 1; i <= length; i=i+1)
11     bar(i, "*");
12 }
```

上のようなプログラムがあったとき、

1. これを実行すると何が表示されるか。
2. メソッド main 中で i が 1 だったときに 11 行目で bar(i, "*"); が実行されたとする。メソッド bar の先頭における i および length の値は何か。
3. さらに進んでメソッド bar のSystem.out.println 実行されたとする。このときの i および length の値は何か。
4. その直後、つまり、main 中の bar(i, "*"); が終了したときの i および length の値は何か。

練習 6-2: (中央揃え)　全ての文字の横幅が一定だと仮定して、n 文字の文字列 m を適当な幅 (w) の中央に位置させて表示させたい。先頭に $(w-n)/2$ 文字の空白を表示した後に m を表示させればよいので、下のような処理になる。

```
1  String message = "Happy 21st Century!"; // メッセージ
2  int width = 50;                         // 表示させる幅
3  int characters = message.length();      // メッセージの文字数
4  // 空白を (width-characters)/2 個表示
5  for (int i = 0; i < (width-characters)/2; i++)
6    System.out.print(' ');
7  System.out.println(message);            // 続けてメッセージを表示
```

1. この処理をメッセージと表示させる幅を引数として受け取るようなメソッドとして定義せよ。
2. 定義したメソッドを使って複数行にわたる適当なメッセージを表示させよ。(例)

```
As soon as we started programming,
 we found to our surprise that it
wasn't as easy to get programs right
        as we had thought.
```

3.3　プログラムの再利用

プログラムをメソッドに分けてゆくと、例えば「中央揃えで表示する」ように、様々なプログラムから利用したくなるメソッドができてくる。Java 言語では別のプログラム (クラス) で定義されたメソッドを使うことができる。方法は、メソッドを使う文

クラス名 . メソッド名 (引数 , 引数 , ...);

の クラス名 の部分を、メソッドが定義されているクラスの名前にするだけである。

例えば、Printing というクラスに、中央揃えで表示 (centered)、右寄せで表示 (flushRight)、横棒を表示 (horizontalBar) というメソッドを図 2 の上のように用意しておき、別のクラスから図 2 の下のようにして使うことができる。

また、同じクラスのメソッドを使う場合は、「 クラス名 . 」の部分を省略できる。例えば、Printing クラスのcentered メソッドが同じPrinting クラスのhorizontalBar メソッドを使っている場合、Printing.horizontalBar(...) と書くかわりに単にhorizontalBar(...) と書くだけでよい。

Java 言語には、数学関数のライブラリはこのような形で提供されている。いままで Math.sin(x) などと書いていたのは、実はMath というクラスに定義されているsin というメソッドを利用するという意味だった。ライブラリがどのような機能をどのような名前で提供しているかは、**API** (application programming interface) と呼ばれる。Java 言語の標準ライブラリの API はマニュアルや web ページ使って調べることができる。

```
1  class Printing {
2    // message を width の幅の中央に表示
3    static void centered(String message, int width) {
4      ...(略)...
5      Printing.horizontalBar(..., " "); // 左側に空白を表示
6      ...(略)...
7    }
8    // message を width の幅の右端にあわせて表示
9    static void flushRight(String message, int width) {
10     ...(略)...
11   }
12   static void horizontalBar(int size, String brick) {
13     ...(略)...
14   }
15 }
```

```
1  class Messenger {
2    public static void main(String[] args) {
3      int size = 60; // 幅 30 文字
4      Printing.horizontalBar("=", size); // 上枠線
5      Printing.centered("Notice");         // タイトル (中央)
6      Printing.flushRight("November 11, 2000"); // 日付 (右寄せ)
7      Printing.horizontalBar("-", size); // 区切り線
8      Printing.centered("...");           // 本文 (中央)
9      Printing.horizontalBar("=", size); // 下枠線
10   }
11 }
```

図 2: 他のクラスのメソッドを使う例: (それぞれ別のファイルに保存し、別々にコンパイルする)

4 答えを返すメソッド

処理手続だけでなく、何らかの答えを持つような計算をメソッドとして定義できれば、複雑な計算過程をブラックボックス化された関数のように扱うことができる。

例えば、最初に示した「何回も登場する平方根の計算の定義を 1 つにまとめたい」場合、標準偏差を求める側では、数行にわたる平方根の計算手順にのかわりに「x の平方根」とだけ書ければ見通しもよくなるわけである。

これを実現するためには、メソッド定義側 (平方根の計算をする側) で計算した結果をメソッドを使う側 (標準偏差を計算する側) に返す必要がある。Java 言語ではreturn 文を使って結果を返すことができる。

4.1 簡単な例

まずは、簡単な例を見よう。預金残高の計算で何度も出てきた「p 円を金利 i で t 年間預けたときの残高の計算」をメソッドとして定義すると下のようになる:

メソッドの定義: いままで違うのは次の 2 点だけである:

返り値の型: 2 行目の メソッドの定義の先頭でstatic double savings(...) の double のように、メソッドが返す答えの型を書く。つまり、このメソッド使う側が受け取る答え (残高) は実数 (double) 型になる。

答えを返す: メソッドから答えを返す場合は 5 行目の return balance; のように return 式; と書く。

```
1  // 元金 balance を年利 interest で term 年預けた場合の残高計算
2  static double savings(double balance, double interest, int term) {
3    for (int year = 1; year <= term; year = year + 1)
4      balance = balance * interest;
5    return balance; // 計算した結果を「使う側」に返す
6  }
7  public static void main(String[] args) {  // 使う側
8    double principal = 100000;  // 元金 10 万円
9    double interest = 1.0045;   // 年利 4.5%
10   int term = 10;              // 期間 10 年
11   // メソッドを使って計算
12   double balance = savings(principal,interest,term);
13   System.out.println(principal+" 円を年利 "+interest+" で "+term
14                      +" 年預けた場合の残高は\n"+balance+" 円です。");
15 }
```

図 3: 答えを返すメソッドの例

メソッドの使用: 12 行目で `double balance = savings(principal,interest,term);` はメソッドを使って計算をしている。計算した結果は、数式などと同様に、変数へ代入等できる。

4.2 標準偏差

値を返すメソッドを使うと、「平方根を計算するメソッド」などを定義できるようになる。「長方形の対角線の標準偏差を計算する」問題も、平方根・2 乗・平均を計算するメソッドを定義することで定義 $\left(\sigma = \sqrt{\overline{(d_i^2)} - \left(\overline{d_i}\right)^2}\right)$ に沿った形でプログラムを作ることができる。

```
1  static double squareRoot(double x) {   // x の平方根を求める
2    ...(略)...
3  }
4  static double square(double x) {return x*x;}  // x の 2 乗を求める
5  static double average(double[] x) { // 数列 x の平均値を求める
6    ...(略)...
7  }
8  static double stdDev(double[] x) {  // 数列 x の標準偏差を求める
9    double squareX[] = new double[x.length]; // 2 乗値の数列を作る
10   for (int i = 0; i < x.length; i=i+1) squareX[i] = square(x[i]);
11   // 標準偏差 = (2 乗値の平均) と (平均値の 2 乗) の差の平方根
12   return squareRoot( average(squareX) - square(average(x)));
13 }
14 public static void main(String[] args) { // ここから実行される
15   // 長方形の幅と高さ
16   double widths[] = new double[] { 20, 18, 24, 31, 17 };
17   double heights[] = new double[] { 4,  5,  7,  2,  3 };
18   // 全ての長方形について対角線長と対角線長の 2 乗を計算
19   double diagonals[] = new double[widths.length]; // 対角線長を
20   for (int i = 0; i < widths.length; i=i+1)       // 三平方定理で求める
21     diagonals[i] = squareRoot(square(widths[i]) + square(heights[i]));
22   // 標準偏差を計算して表示
23   System.out.println("対角線長標準偏差は"+stdDev(diagonals)+"です。");
24 }
```

このプログラムでは、18–21 行目が $d_i = \sqrt{w_i^2 + d_i^2}$ を計算している部分。また、メソッド`stdDev` 中では、まず、d_i^2 を計算しておき、それから平均値を求めるメソッド`average` を用いて $\sigma = \sqrt{\overline{(d_i^2)} - \left(\overline{d_i}\right)^2}$

を計算している。

練習 6-3: 閏年　ある西暦年が閏年のとき true、そうでないときはfalse と答えるメソッド isLeap を定義せよ。isLeap を使って、1900 年、1996 年、1999 年、2000 年、2001 年 が閏年かどうかを表示させるプログラムを作れ。

練習 6-4: 月の日数　西暦年と月が与えられたとき、その月の日数を答えるメソッドmonthDays を定義せよ。monthDays を使って 2000 年の各月の日数を表示させるプログラムを作れ。

練習 6-5: 曜日を調べる　与えられた年月日の曜日を計算するには、過去の 1 日 (例えば 1900 年 1 月 1 日の月曜日) から与えられた日まで 1 日ずつ曜日を進めてゆけばよい。曜日を (日曜日=0, 月曜日=1, ...) のようにして表わすことにすると、以下のような繰り返しによって計算できる。

```
// 目標とする年月日に達するまで曜日を増やし続ける
while ((year < goalYear) || (month < goalMonth) || (day < goalDay)) {
  dayOfWeek = (dayOfWeek + 1) % 7; // 曜日を増やす
  day = day + 1;                   // 日を増やす
  if (day > monthDays(year,month)) { // 月末を越えた場合
    day = 1;                         // 1 日に戻って
    month = month + 1;               // 月を増やす
    if (month > 12) {   // 年末を越えた場合
      month = 1;        // 1 月に戻って
      year = year + 1;  // 年を増やす
    }
  }
}
```

与えられた年月日の曜日を数字で答えるメソッド dayOfWeek を定義せよ。これを用いて、自分の生まれた日の曜日、来年の自分の誕生日の曜日を調べるプログラムを作れ。

練習 6-6: カレンダ　与えられた年月のカレンダを表示するプログラムを作れ。(例)

```
   November 2000
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30
```

(ヒント: 整数の桁数を計算するメソッドや、整数を指定された幅で表示するメソッドを定義しておくとよい。)