計算機プログラミング I (10)

オブジェクト指向プログラミング

2000 年 12 月 12 日 増原 英彦

1 データと処理の一体化

1.1 オブジェクトとメソッドを使った問題解決

プログラミングとは、与えられた問題を解く手順を考え、それをプログラムにすることである。現実の問題を解く場合には、現実世界にあるモノを、プログラム中のデータに対応させて操作する方法が有効である。

前回学んだオブジェクトを用いると、複数の値から成る情報を 1 つの値として扱うことができるため、現実世界のモノを表現するのに適している。例えば、ロールプレイングゲーム (RPG) を作ることを考えてみる。ゲーム (現実) の世界には「人間」や「怪物」がいて、それぞれ「現在位置」「体力」「持ち物」「攻撃力」などの情報を持っている。オブジェクトを使えば、プログラム中でも「人間」や「怪物」といった単位の操作を書くことができる。

問題解決の手順とは、モノに対する操作が並んだものになる。その際、細々とした一連の操作には、適切な名前をつけて抽象化することで、見通しよく手順を考えることができる。上の RPG の例であれば、ゲーム全体は (1)「人間や怪物を少し移動させる」 (2) 「怪物が近くにいる人間を攻撃する」 (3)「人間が近くにいる怪物を攻撃する」といった手順から成っていて、「人間を少し移動させる」手順は (a) 「現在の位置を調べる」 (b) 「体力と周囲の地形から移動距離を計算する」(c) 「新しい位置を計算する」 (d)「画面上の絵を移動させる」という細かな手順から成っている、といった具合である。

すでに学んだように、プログラミングにおいては細かな手順をメソッドとしてまとめ、名前をつけることができる。おおまかな手順を書くときには、そのメソッドの名前を書くだけで具体的な手順を気にせずに操作することができる。

1.2 対象による手順の違い

ある程度大きな問題を扱おうとすると、

同じ名前の操作でも、対象によって具体的な手順が違う

ことが起きる。例えば「移動させる」という操作でも、対象が「人間」であれば上記のような手順になる。一方、「魔法の使える怪物」であれば、周囲の地形を無視して遠くに移動するかも知れない。また、夜しか移動できない「怪物」もいるかも知れない。しかし、おおまかな手順を考える際には、それらの細かな手順の差異は気にせずに単に「移動する」という名前で呼んでいるわけである。

そのためプログラムとしては (1) 対象ごとに「移動」メソッドをそれぞれ定義して、(2) おおまかな手順をプログラムするときは、対象によって適切な「移動」メソッドを使い分けなければならない。

1.3 オブジェクト指向

Java 言語をはじめとするオブジェクト指向プログラミング言語では、オブジェクトを操作の対象とするメソッドを定義することができる。このメソッドを同じ名前 (例えば「移動」) で異なるクラス (「人間」と「怪物」) に定義しておくと、オブジェクトを操作する側がその名前のメソッドを使うだけで、自動的に操作の対象となるオブジェクトのクラスに応じたメソッドが実行される。これを使えば、同じ名前を使いつつ、対象によって異なる処理をすることができるようになる。別の言い方をすれば、各オブジェクトが、自分に対する操作の具体的な手順を知っていて、操作する側は処理内容を共通の名前で依頼しているとも言える。

2 インスタンスメソッド

オブジェクトを操作の対象としたメソッドの書き方、使い方を見てゆこう。例として 2 次元平面上の「点」を考える。「点」に対する操作には

- 新しく点を作る
- 点の x, y 座標を調べる
- 点を原点から定数倍の位置へ移動させる
- 文字列に変換する
- 点を (dx, dy) だけ移動する
- 点を原点まわりに回転させる

といったものが考えられる。

オブジェクトに対する操作を定義する方法であるインスタンスメソッド[1]を使って、上の操作を書いてみよると、図 1 のようになる。この場合、`Point` クラスに対する操作は`Point` クラスの中に書かなければいけないことに注意せよ。

- オブジェクトを作る式は`new Point(10,20)` のように (図 1 下-3 行目) 引数を与えることができる。ここでは「最初の位置が (10,20) である点」を作っている。
- 上のように引数付きでオブジェクトを作ると、オブジェクト作られた後にコンストラクタと呼ばれる特別なメソッドが実行される。コンストラクタは

  クラス名 (引数 , ...) {...}

  という形で書く。図 1 上では 4 行目の`Point(int x, int y)` {...} がコンスラクタである。
- コンストラクタとインスタンスメソッドの中では、`this.x` のように `this` という特別な変数を通して「現在操作の対象となっているオブジェクト」を扱うことができる。図 1 上-4 行目では、「現在操作の対象となっているオブジェクトのインスタンス変数`x` に、引数として受け取った`x` の値を代入」している。(`y` も同様。)

[1] 講義第 6 回では、一般的な処理手順をまとめるメソッドを学んでいる。これは、インスタンスメソッドと対比してクラスメソッドと呼ばれたり、`static` メソッドと呼ばれたりする。

```
1  class Point { // 2次元平面の点 (Point.java というファイルに保存)
2    int x, y;                        // 座標値 (インスタンス変数)
3    // コンストラクタ: 与えられた座標値上の点を作る
4    Point(int x, int y) {this.x = x; this.y = y;}
5    // 以下がインスタンスメソッド
6    int getX() { return this.x; } // x 座標値を調べる
7    int getY() { return this.y; } // y 座標値を調べる
8    void scale(double factor) {   // 点を原点から factor 倍のところへ移動させる
9      this.x = (int)(this.x * factor); this.y = (int)(this.y * factor);
10   }
11   // 点の位置を「(x,y)」という形式の文字列に変換する
12   public String toString() { return "("+this.x+","+this.y+")";}
13 }
```

```
1  class PointUser { // Point を使うプログラム (PointUser.java に保存)
2    public static void main(String[] args) {
3      Point p1 = new Point(10,20); // 点を (10,20) に作る
4      System.out.println(p1.getX()); // 点の x 座標を表示
5      p1.scale(0.5);               // p1 を半分にする
6      System.out.println(p1.toString()); // 点を表示
7    }
8  }
```

図 1: 点オブジェクトの定義とそれを使う例

- インスタンスメソッドを使う式は`p1.getX()`(図 1 下-4 行目) のように、

  オブジェクト.メソッド名(引数)

  という形式で書く。このように書くと、そのオブジェクト (`p1`) のクラスで定義されているメソッド (`getX`) が選ばれて使われる。

練習 10-1: (移動) 「点」を与えられたベクトル量だけ移動させる`void move(Point dp) {...}` のようなメソッドを`Point` クラスに追加せよ。ただし、`dp` は移動量を与えるベクトルだとする。

練習 10-2: (回転) 「点」を原点のまわりに回転させる`void rotate(double degree) {...}` のようなメソッドを`Point` クラスに追加せよ。ただし、`degree` は回転角 (360 度で 1 回転) だとする。
(※ 点 (x, y) を原点まわりに回転させるた座標は、$(x\cos\theta - y\sin\theta, x\sin\theta + y\cos\theta)$ である。ただし、Java 言語の三角関数`Math.cos`, `Math.sin` は角度をラジアン (2π で 1 回転) で与えることに注意。)

練習 10-3: (点の描画) 「点」の位置に小さな丸をグラフィクス画面上に表示させる`void draw(java.awt.Graphics g) {...}` のようなメソッドを `Point` クラスに追加せよ。ただし、`java.awt.Graphics` は Java 言語標準のグラフィクス描面用のクラスで、`g.fillOval(x,y,width,height);` のようにすると左上の位置が `x, y` で幅 `width` 高さ `hegiht` の楕円を描くようなメソッド等が用意されている[2]。

[2]他にどのようなメソッドがあるかは、講義の web ページから、「参考情報」の「Java 言語に関する情報」をたどり、「Java 言語公式説明書」の中の「Java プラットフォーム Core API」「パッケージ java.awt」「Graphics」をたどって見るとよい。

また、簡単にグラフィクス画面を作るためには、講義で用意したDrawer2というクラス (定義はwebページにある) を使って、例えば下のようにする。

```
java.awt.Graphics g = Drawer2.createFrame(200,200);
g.fillOval(150,150,100,100);
```

練習 **10-4:** (質点系)　放物運動をする質点の軌跡を表示するプログラムを以下のようにして作れ:

- 質点の位置・速度・加速度を表わす「点」オブジェクトをそれぞれ作る
- 各ステップで以下の操作を行い、それを適当な回数だけ繰り返す
  1. 質点の位置に小さな円を描く
  2. 質点の位置を速度ベクトルの量だけ移動させる
  3. 速度ベクトルを加速度ベクトルの量だけ変化させる

ただし、質点の最初の位置と速度は $(0, 200)$ と $(10, -20)$ であり、加速度は常に $(0, 4)$ であるとする。

3　同名のメソッド

「点」によく似たデータとして「直線」を考えてみよう。「直線」は位置情報として 2 つの「端点」を Point クラスのオブジェクトとして持つ。x, y 座標を求める場合は、直線の中心の座標を計算することにする。それ以外に、回転・移動などは Point クラスと同様のものを持つとする。クラス定義と、それを使う例は下のようになる:

```
 1  class Line {                      // 直線
 2    Point corner1, corner2;         // 両端点 (インスタンス変数)
 3    // コンストラクタ
 4    Line(Point corner1, Point corner2) {
 5      this.corner1 = corner1; this.corner2 = corner2;
 6    }
 7    // インスタンスメソッド
 8    // x,y 座標を求める — 中点の座標とする
 9    int getX() { return (this.corner1.getX()+this.corner2.getX())/2; }
10    int getY() { return (this.corner1.getY()+this.corner2.getY())/2; }
11    void scale(double factor) {   // 原点から定数倍
12      this.corner1.scale(factor); // 両端点を定数倍する
13      this.corner2.scale(factor);
14    }
15    public String toString() {    // 文字列に変換
16      return this.corner1.toString() + "-" + this.corner2.toString();
17    }
18  }
```

```
1  class LineUser {                  // Line を使うプログラムの例
2    public static void main(String[] args) {
3      Point p1 = new Point(10,20); // (10,20) の点を作る
4      Point p2 = new Point(40,30); // (40,30) の点を作る
5      Line l = new Line(p1,p2);
6      System.out.println(l.getX()); // x 座標を表示
7      l.scale(0.5);                 // 半分にする
8      System.out.println(l.toString()); // 表示
9    }
10 }
```

LineUser の 6–8 は行目は、図 1 下の 4–6 行目とほとんど同じである。つまり、これらのプログラム中の getX, scale などのメソッドは同じ名前であっても、p1 や l のクラスによって実行されるメソッドが選ばれている。

練習 **10-5:** (線の描画)　練習 10-3 にならって、「線」を画面上に描画するメソッドを追加せよ。

4　タートル・グラフィクス

より複雑な例として、「亀」を回転させたり歩かせたりして、その足跡で絵を描く「タートル・グラフィクス」を考えてみよう。

1 匹の「亀」は、次のような情報を持っている:

- 画面上の位置
- 現在頭が向いている角度
- 足跡を描く画面

また、「亀」に対する操作としては、次のようなものが考えられる:

- 現在頭の向いている方向へ与えられた距離だけ歩き、足跡を画面上に残す (walk)
- 頭の向いている方向を回転させる (turn)

Point クラスを使って「亀」の位置を表わせば、「位置を動かす」「ベクトルを回転させる」といった操作の具体的な内容は気にせずに「亀」に対する操作を定義することができる。

```
 1  class Turtle {// 亀の軌跡で絵を描く(Turtle.java というファイルに保存)
 2    Point position;               // 現在亀がいる場所
 3    double angle;                 // 亀の頭がむいている角度(360 度で 1 回転)
 4    java.awt.Graphics g;          // 軌跡を描く画面
 5
 6    Turtle(java.awt.Graphics g) {  // コンストラクタ
 7      this.position = new Point(0,0); // 最初の場所は原点
 8      this.angle = 0;             // 最初の角度は 0
 9      this.g = g;                 // 画面は作る側から受け取る
10    }
11
12    void walk(int distance) {     // 亀を指定された距離だけ移動させ、軌跡を描く
13      // 移動前の x,y 座標を覚えておく
14      int startX = this.position.getX(), startY = this.position.getY();
15      // 移動させる
16      Point dp = new Point(distance,0); // 移動距離のベクトルを作る
17      dp.rotate(this.angle);      // ベクトルを現在頭が向いている角度へ回転
18      this.position.move(dp);     // 現在位置にベクトルを足す
19      // 移動後の x,y 座標を調べる
20      int endX = this.position.getX(), endY = this.position.getY();
21      g.drawLine(startX,startY,endX,endY); // 移動前後の位置間に線を引く
22    }
23
24    void turn(double dAngle) {    // 亀の角度を変える
25      this.angle = this.angle + dAngle;
26    }
27  }
```

定義された「亀」を使うことで様々な絵を描くことができる。例えば三角形を描くには、「移動して回転する」ことを 3 回繰り返せばよいので、下のようになる:

```
1  (軌跡を表示する画面 g を作る—略)
2  Turtle t = new Turtle(g);   // 亀を作る
3  (t を画面の中央上方に移動させる—略)
4  int length = 150;           // 辺の長さ
5  for (int edge=1; edge<=3; edge=edge+1) { // 辺を 3 つ描く
6    t.walk(length);           // 辺の長さ分移動させ
7    t.turn(120);              // 120 度回転させる
8  }
```

(完全版は web ページ)

練習 **10-6:** (飛ぶ亀) 「t を画面の中央上方に移動させる」ような場合に、軌跡を残さずに移動させることができると、より見栄えがするだろう。そこで、そのようなメソッドとして `void jump(int distance)` $\{\ldots\}$ をクラスTurtle に追加せよ。メソッド `jump` とメソッド `walk` は大部分が同じになるので、共通する部分をひとつのメソッドにまとめるとよい。

練習 **10-7:** (多角形) 三角形だけでなく、n 角形を描くプログラムを作ってみよ。また、星形のような図形を描くプログラムも作ってみよ。

練習 **10-8:** (木) 長さ ℓ の幹を持つ木を「亀」を使って描く手順は、

- 長さ ℓ だけ亀を動かす
- 左に適当な角度 (例えば 30 度) 回転して、一まわり小さい (例えば 0.85ℓ) 長さの幹を持つ木を描く
- 右に適当な角度回転して、やはり一まわり小さい長さの幹を持つ木を描く
- 最初の位置に戻る

のように、再帰的に定義でき、以下のようなメソッドになる (ここではTurtle クラスにインスタンスメソッドを追加する場合の例を示す):

```
 1  void tree(int length) {       // 木を描く (クラス Turtle に追加)
 2    int minimumLength = 5;      // 最短の幹の長さ
 3    int branchAngle = 30;       // 技分かれの角度
 4    double decayFactor = 0.85; // 幹を短くする割合
 5    if (minimumLength <= length) {
 6      this.walk(length);        // 幹を描く
 7      this.turn(-branchAngle); // 左に回転
 8      tree((int)(length*decayFactor)); // 一まわり小さい木を描く
 9      this.turn(2*branchAngle);// 右に回転
10      tree((int)(length*decayFactor)); // 一まわり小さい木を描く
11      this.turn(-branchAngle); // 頭の向きを戻して
12      this.jump(-length);       // 出発点に戻る
13    }
14  }
```

このプログラムを完成させて木を描いてみよ。また、技分かれの角度や短くする割合を変化させて、できあがる木の形の変化を見てみよ。(乱数を使うと、自然な木らしくなる。)

A クラスメソッドによる定義

図 1 でインスタンスメソッドを用いて定義したPoint クラスに対する操作を、クラスメソッドによって定義した場合を示す。

```
 1  // 点に対する操作 (PointOperations.java というファイルに保存)
 2  class PointOperations {
 3    static Point make(int x, int y) { // 与えられた座標値上の点を作る
 4      Point p = new Point();
 5      p.x = x; p.y = y;
 6      return p;
 7    }
 8    static int getX(Point p) { return p.x; } // x 座標値を調べる
 9    static int getY(Point p) { return p.y; } // y 座標値を調べる
10    static void scale(Point p, double factor) { // 原点から factor 倍
11      p.x = (int)(p.x * factor); p.y = (int)(p.y * factor);
12    }
13    // 点の位置を「(x,y)」という形式の文字列に変換する
14    static String toString(Point p) { return "("+p.x+","+p.y+")";}
15  }
```

```
1  class PointUser {                // Point を使うプログラムの例
2    public static void main(String[] args) {
3      Point p1 = PointOperations.make(10,20); // p1 を作る
4      System.out.println(PointOperations.getX(p1)); // p1 を表示
5      PointOperations.scale(p1,0.5); // p1 を半分にする
6      System.out.println(PointOperations.toString(p1)); // p1 を表示
7    }
8  }
```