計算機プログラミング I (11)

継承

2000 年 12 月 19 日 増原 英彦

オブジェクト指向プログラミングには、様々な側面がある。前回学んだデータと操作手順 (メソッド) をまとめる機能はその一つである。今回は、以前作ったプログラムに機能を追加したり、動きを変更するための継承(inheritance) 機能を紹介する。

1 機能の追加と変更

大きなプログラムは簡単には作れない。できるだけ簡単に作るために、小さなプログラムを分担して作りそれを後から組み合わせることや、昔作ったプログラムを手直しして部分的に再利用することなどが行われている。

ところが小さなプログラムを分担して作ると、「よく似ているけどちょっと違うことをする」プログラムが沢山作られてしまいがちである。例えば、ロールプレイングゲームを作ることを考えよう。ゲームに登場する様々な「怪物」をそれぞれ分担して作ると、「怪物」に共通する性質や操作が沢山あるのに、個別に同じようなクラス定義やメソッド定義を書いていることが起きるかも知れない。結果として冗長な仕事が増えてしまい、労力はあまり減らないかも知れない。

また、昔作ったプログラムを手直しする一つの方法は、ソースプログラムをコピーして、必要な場所を書き換えることだろう。しかしこのやり方は、大量のコピーができるため様々な混乱を引き起こす。

継承を使うと、すでに定義したクラスに新しい機能や性質を加えたクラスを作ることができる。これを使えば、「怪物」に共通するクラス定義を 1 つ作っておき、あとは個々の「怪物」に特有の操作や性質だけを定義できるので、労力を減らすことができるだろう。

例として、「点」を表わすクラスから「色付きの点」をクラスを作るものを見てみよう。まず、「点」クラスとして図 1 のようなものが定義されているとする。この「点」は x, y 座標[1]というインスタンス変数と、いくつかのメソッドを持っている。例えばdraw メソッドは、点のある位置に (現在の色で) 小円を描くものとする。

この「点」に「色」情報[2]を加えたクラスを作ることにしよう。この「色付きの点」は、移動 (move) や回転 (rotate) といった操作は「点」クラスと同じ内容であり、描画 (draw) だけ少し違った操作をすることになるが、クラス定義の継承を使うことによって、どのように定義できるかを見てみよう。

2 インスタンス変数・メソッドの追加

図 2 (上) は、Point クラスに「色」を表わすインスタンス変数と「色付きで小円を描く」メソッドを追加したクラスである。また、図 2 (下) は、この「色付きの点」を使って物体の放物運動を表示し

[1]前回は整数型を使っていたが、今回は実数型にしているが、本質的な違いではない。

[2]Java 言語のグラフィクス画面上の色を表わすには、java.awt.Color というクラスのオブジェクトを使う。

```
 1  class Point {                        // 点クラス
 2    double x, y;                       // インスタンス変数
 3    // 2 つの実数から、それを x,y 座標とするような点を作るコンストラクタ
 4    Point(double x, double y) { this.x = x; this.y = y; }
 5    // インスタンスメソッド
 6    double getX() { return this.x; }   // 点の x 座標を調べる
 7    double getY() { return this.y; }   // 点の y 座標を調べる
 8    ... 他に move,rotate,scale,toString,length などのメソッド (略)...
 9    // グラフィクス画面 g 上に点を描く or 消す
10    void draw(java.awt.Graphics g) {   // 点を描く
11      int size=3*2;                    // 小円の直径
12      int x=(int)(this.getX()-size/2), y=(int)(this.getY()-size/2);
13      g.fillOval(x,y,size,size);       // fillOval で円を描く
14    }
15  }
```

図 1: 「点」のクラス定義

ているプログラムである。

- 図 2 (上) の 1 行目のclass ColorPoint1 extends Point のように、extends というキーワードをクラス名の後ろにつけることで、あるクラス (Point) を継承したクラスを定義できる。このとき、Point を親クラス、ColorPoint1 を子クラスあるいはサブクラスと言う。
- 子クラスには、インスタンス変数やメソッドを定義できる。
- 子クラスのコンストラクタ (図 2 (上)4–7 行目ColorPoint1(int x, ...)) の中では親クラスのコンストラクタを実行しなければならない。上の定義のようにsuper(x,y);(5 行目) と書くと、下のような順序でオブジェクトが作られる。
  1. 図 2 (下) の 2 行目のnew ColorPoint1(0,200,java.awt.Color.red); のようにしてオブジェクトを作ると、ColorPoint1 のオブジェクトが作られる
  2. 子クラスのコンストラクタ (ColorPoint1(int x, ...)) が実行される
  3. super(x,y); が実行され、親クラスのコンストラクタ (図 1 の 4 行目 Point(int x, int y)) が実行される
  4. 図 2 の 6 行目のthis.c = c; によってオブジェクトのインスタンス変数c にjava.awt.Color.red が代入される
- 子クラスのオブジェクトは、親クラスの全てのインスタンス変数およびメソッドを持っている。

  図 2 (下) のように、ColorPoint1 クラスのオブジェクト (point1) を作った場合、point1.move(velocity1); のようにPoint クラスに定義されていたメソッドがそのまま使える。

練習 **11-1:** (色付きの亀) 前回の 4 章で示したタートル・グラフィクスの Turtle クラスを継承して、色付きの亀のクラスを作れ。

練習 **11-2:** (円) 「点」に「半径」という情報を加えれば「円」を表わすことができる。「円」を表わすクラスをPoint クラスの子クラスして定義し、円を描く drawCircle というメソッドを追加せよ。

```
1  class ColorPoint1 extends Point { // 色付きの点
2    java.awt.Color c;              // 点の色 (追加)
3    // コンストラクタ
4    ColorPoint1(int x, int y, java.awt.Color c) {
5      super(x,y);                  // Point のコンストラクタを使う
6      this.c = c;                  // 座標値と色を覚える (追加)
7    }
8    void drawWithColor(java.awt.Graphics g) { // 色付円を描く
9      g.setColor(c);               // 色を設定してから (追加)
10     int size=3*2;                // 小円の直径 (以下 Point と同じ)
11     int x=(int)(this.getX()-size/2), y=(int)(this.getY()-size/2);
12     g.fillOval(x,y,size,size);  // fillOval で円を描く
13   }
14 }
```

```
1  // 色付きの点を使って 2 つの点の放物運動を表示する
2  ColorPoint1 point1 = new ColorPoint1(0,200,java.awt.Color.red);
3  ColorPoint1 point2 = new ColorPoint1(0,200,java.awt.Color.blue);
4  Point velocity1 = new Point(10, -20); // 各点の速度
5  Point velocity2 = new Point(15, -30);
6  Point accel = new Point(0,4); // 加速度 (共通)
7
8  int steps = 20;          // 20 ステップだけ点を動かす
9  for (int i = 0; i < steps; i= i+1) {
10   point1.drawWithColor(g); point2.drawWithColor(g);     // 表示
11   point1.move(velocity1);  point2.move(velocity2); // 移動
12   velocity1.move(accel);   velocity2.move(accel); // 速度を変化
13 }
```

図 2: 色付きの点クラス (上) とそれを使ったプログラム (下)

3　メソッドの上書き

上で示したColorPoint1 クラスでは、色付きの表示をするときに新しい名前のメソッド (drawWidthColor) を定義していた。これを図 3 (上) のように、親クラス (Point) で定義されていたものと同じ名前 (draw) のメソッドを子クラスに定義できる。

このようにして定義された子クラスのオブジェクトを、親クラスのオブジェクトを扱うプログラムに与えて使うことができる。このとき、上書きされたメソッドは、子クラスのものが使われるため、親クラスと少し動作が違うオブジェクトができることになる。

このようにすると、Point クラスのために書かれたプログラムに、ColorPoint2 クラスのオブジェクトを作って渡すだけで、色付きの点が描かれることになる。図 3 (下) のプログラムでは、

- 1–10 行目のメソッドは、グラフィック画面、物体の位置、速度、加速度を受け取り、物体の運動を表示するメソッドである。このプログラムは親クラスであるPoint クラスのオブジェクトを受け取って計算するように作られている。

- 15–17 行目では、位置、速度、加速度をPoint クラスのオブジェクトとして作り、18 行目でメソッドを使っている。これはいままで通りである。

- 21 行目では、位置として Point クラスの子クラスであるColorPoint2 クラスのオブジェクトを作り、それをPoint クラスの型を持つ変数point2 に代入している。

  これは一見、変数の型と値の型が違っているように見えるが、子クラス (ColorPoint2) のオブジェクトは親クラス (Point) のオブジェクトとして見なすことができ、正しく代入できる。

- 23 行目では、メソッドparabola に上で作ったColorPoint2 クラスのオブジェクトを渡している。メソッドの引数はPoint クラスの型になっているが、代入と同様、子クラスのオブジェクトを渡すことができる。

- 6 行目では、position という変数が指すオブジェクトのdraw というメソッドを実行している。ここで、変数の型はPoint クラスであるが、変数が指しているオブジェクトはColorPoint2 クラスのオブジェクトであるので、ColorPoint2 クラスのdraw メソッドが実行される。

  つまり、メソッドを実行する際は、そのときのオブジェクトがどのクラスのオブジェクトであるかによって決まるわけである。

4　親クラスのメソッドを使う

上のメソッドdraw では、「色を付けて」「小円を描く」操作という 2 つのものに分けられる。ところが後者の操作はPoint クラスのメソッドdraw と全く同じである。

メソッドの中でsuper という変数を使うと、親クラスのメソッドを実行することができる。下の例では、メソッドdraw の中で色を設定してからsuper.draw(g); とすることで、Point クラスのdraw メソッドを実行する。

練習 11-3: (直線)　「直線」を表わすオブジェクトは、「直線の始点」と「始点から終点へのベクトル」の情報を持っていると考えることができ、「点」に性質を追加したものだと言える。

```
class ColorPoint2 extends Point { // 色付きの点
  java.awt.Color c;              // 点の色 (追加)
  // コンストラクタ
  ColorPoint2(int x, int y, java.awt.Color c) {
    super(x,y);                  // Point のコンストラクタを使う
    this.c = c;                  // 座標値と色を覚える (追加)
  }
  void draw(java.awt.Graphics g) { // 色付円を描く
    g.setColor(c);               // 色を設定してから (追加)
    int size=3*2;                // 小円の直径 (以下 Point と同じ)
    int x=(int)(this.getX()-size/2), y=(int)(this.getY()-size/2);
    g.fillOval(x,y,size,size);  // fillOval で円を描く
  }
}
```

```
// 「点」の放物運動を表示するメソッド
static void parabola(java.awt.Graphics g,
                Point position, Point velocity, Point accel) {
  int steps = 20;                // 20 ステップだけ点を動かす
  for (int i = 0; i < steps; i=i+1) {
    position.draw(g);            // 表示
    position.move(velocity);     // 移動
    velocity.move(accel);        // 速度を変化
  }
}

public static void main(String[] args) {
  ...(略)...
  // 普通の「点」を作る
  Point point1 = new Point(0,200);
  Point velocity1 = new Point(10, -20); // 点の速度
  Point accel = new Point(0,4);          // 加速度 (共通)
  parabola(g,point1,velocity1,accel);   // 放物運動を描く

  // 「色付き点」のクラスを使って青い「点」を作る
  Point point2 = new ColorPoint2(0,200,java.awt.Color.blue);
  Point velocity2 = new Point(15, -30); // 点の速度
  parabola(g,point2,velocity2,accel);   // 放物運動を描く
}
```

図 3: メソッドを上書きする例とそれを使うプログラム

```
1  class ColorPoint3 extends Point { // 色付きの点
2    java.awt.Color c;              // 点の色
3    // コンストラクタ
4    ColorPoint3(int x, int y, java.awt.Color c) {
5      super(x,y);                  // 親クラスのコンストラクタ
6      this.c = c;                  // 色を覚える
7    }
8    void draw(java.awt.Graphics g) {
9      g.setColor(c);               // 色を設定してから
10     super.draw(g);               // 親クラスの draw を使って小円を描く
11   }
12 }
```

図 4: 親クラスのメソッドを使う例

1. この考え方に基いて「直線」クラスを定義し、どのようなメソッドは親クラスのものを再利用でき、どのようなものは「上書き」しなければいけないかを確認せよ。

2. また、放物運動を表示するプログラムの中で、質点を表わすのに「直線」オブジェクトを用いても、プログラムがそのまま動作することを確めよ。

3. 直線の長さを計算するメソッド `width` を追加せよ。

練習 11-4: (四角形) 「四角形」は「幅のある直線」と考えることができる。つまり、「直線」に「幅」という性質を加えたものだと言える。

1. この考え方に基いて「四角形」クラスを定義せよ。

2. また、放物運動を表示するプログラムの中で、質点を表わすのに「四角形」オブジェクトを用いても、プログラムがそのまま動作することを確めよ。

練習 11-5: (複合図形) ここまでの練習で「円」「直線」「四角形」は、全て「点」の子クラスとして定義してきた。「円」と「直線」を組み合わせた「串だんご」のような図形や 3 つの「直線」から成る「三角形」を、1 つのデータとして表わしたい。このプログラムの概略は以下のようになるだろう:

```
1  class Compound {                // 複合図形クラス
2    Point[] elements;             // 図形要素の配列
3    Compound(Point[] elements) {this.elements = elements;} // コンストラ
   クタ
4    void move(Point dp) {         // 図形を移動する
5      for (int i = 0; i < elements.length; i=i+1) // 各図形要素を
6        elements[i].move(dp);                     // 移動するだけ
7    }
8    void draw(java.awt.Graphics g) { // 図形を表示する
9      for (int i = 0; i < elements.length; i=i+1) // 各図形要素を
10       elements[i].draw(g);                      // 表示するだけ
11   }
12   ... 以下略...
13 }
```

これを使うプログラムは例えば下のようになる:

```
1  Point[] elements =                                    // 図形要素の配列を作る
2    new Point[] { new Circle(10,190,10),                // 円と
3                  new Line(0,180,20,200),               // 直線と
4                  new Rectangle(0,180,20,200) };  // 正方形で構成
5  Compound p = new Compound(elements); // 複合図形を作る
6  Point velocity = new Point(10, -20); // 速度
7  Point accel = new Point(0,4); // 加速度
8
9  java.awt.Graphics g = Drawer2.createFrame(200, 200);
10 int steps = 20;          // 20 ステップだけ動かす
11 for (int i = 0; i < steps; i= i+1) {
12   p.draw(g);            // 図形を表示し
13   p.move(velocity1);    // 移動させ
14   velocity1.move(accel);        // 速度を更新する
15 }
```

このプログラムを完成させ、複合オブジェクトも「点」と同じように扱えることを確認せよ。

練習 **11-6:** (距離を測る亀)　前回の講義で作成した亀 (タートルグラフィクス) をもとに、「歩いた距離の合計」を覚えるものを以下のようにして作れ。

1. 前回のクラス `Turtle` を継承したクラスを作る
2. そのクラスに合計距離を覚えるインスタンス変数を追加する
3. 「歩く」メソッドを上書きして、合計距離を加算するように変更する
4. 「合計距離を求める」「合計距離をゼロに戻す」メソッドを追加する

この亀を使って、「三角形」や「木」の線の長さを計算させてみよ。

練習 **11-7:** (酔った亀)　前回の講義で作成した亀 (タートルグラフィクス) をもとに、「回転するときの角度」や「歩くときの距離」が、毎回乱数によって少しだけ大きかったり小さかったりするものを作れ。継承によって親クラスのメソッドを使うことで、どれだけ定義が簡単にできるかを見てみよ。

なお、`Math.random()` という式を計算すると、0 以上 1 未満の乱数が得られる。

5　継承のまとめ

5.1　文法

クラス定義:　クラス定義の際に親クラスを指定する。

```
class [クラス名] extends [親クラス名]
{
  [インスタンス変数やメソッドの定義]
}
```

親クラスのメソッドを使う: インスタンスメソッドの中で、親クラスで定義されたメソッドを使う場合には、super という特別な変数を使う:

super.メソッド名(引数, ...)

(super は、現在操作の対象となっているオブジェクトを親クラスのオブジェクトとして見たものと言える。)

親クラスのコンストラクタを使う: コンストラクタの中で、親クラスのコンストラクタを使う場合にもsuper という特別な名前を使う:

super(引数, ...);

5.2 型の関係

```
class A extends B { ... }
```

のようにクラスが定義されているとき、クラスA のオブジェクトはクラスB のオブジェクトとして使える。つまり、クラスB 型の変数にクラスA のオブジェクトを代入することができたり、クラスB 型の引数をとるメソッドにクラスA のオブジェクトを引数として渡すことができる。

5.3 メソッド実行

オブジェクト式.メソッド名(引数,...)

のようなメソッドを使う式があったとき、実際に実行されるメソッドは オブジェクト式 が実際に指し示しているオブジェクトのクラスから順に親クラスをたどり、最初に メソッド名 という名前で定義されていたメソッドが使われる。プログラム的に書くと次のようになる:

```
o = (オブジェクト式が指し示すオブジェクト);
c = (o のクラス);
while (c に メソッド名という名前のメソッドは定義されていない)
  c = (c の親クラス);
クラス c の メソッド名 という名前のメソッドを実行;
```

あるクラスのメソッド定義の中でsuper.メソッド名(引数,...) のような式があったときは、そのクラスの親クラスから順に親クラスをたどり、最初に メソッド名 という名前で定義されていたメソッドが使われる。