数理情報セミナー・第 4 回

2016 年 10 月 13 日

今回の内容

4.1 クラス宣言 4–1
4.2 インスタンス変数の宣言 4–3
4.3 コンストラクタの宣言 4–4
4.4 this 4–5
4.5 インスタンスメソッドの宣言 4–6
4.6 演習問題 4–8

4.1 クラス宣言

前回までのプログラムは、この科目のクラスライブラリが提供している GameFrame、Card、Deck などのクラスを利用しているだけでした。必要に応じて、これらのクラスのインスタンスを生成し、生成したインスタンスにいろいろな仕事を依頼することで全体のプログラムを機能させてきましたが、これらのクラスの使い方を知る必要はあっても、これらのクラスがどのようなプログラムとして実現されているのかはとくに気にしていませんでした[1]。

非常に単純なプログラムなら、既存のクラスを利用するだけで十分な場合もありますが、少し大きなプログラムを作成する場合は、必要なクラスを自分でいくつも定義して、そのクラスを利用して全体のプログラムを実現することになります。今回は、そのような場合に必要となるクラスの定義方法について勉強します。

第 1 回に説明したように、Java では、そのクラスのオブジェクトがどのようなものであるのかを**クラス宣言**という書式で記述します。これがクラスの定義に相当します。クラス宣言の最も単純な書式は次のようなものです。

```
class [クラス名] {
        [インスタンス変数、コンストラクタ、インスタンスメソッドなどの宣言]
}
```

クラス宣言では、class というキーワードの前に、そのクラスの修飾子[2]として、public、final、abstract などのキーワードを置くこともあります[3]。[クラス名] は、自分で名付けたクラスの名前です。クラス名は、Java のキーワードや null、false、true などと重ならないように自由に選ぶことができます[4]が、Card や GameFrame のように、各単語の先頭のみを英大文字として残りを英小文字とするのが慣習ですので、これに従いましょう。[クラス名] と { の間には、extends [クラス名] や implements [インタフェース名の列] のような書式が挟まることもあります[5]。

[1] このことは、カプセル化の観点からは自然なことであり、望ましいことです。

[2] プログラミング言語の中で、そこに宣言されているクラスや変数やメソッドなどの性質を指定する (形容詞的な) 働きを持つものを、一般に修飾子と呼びます。

[3] これらの修飾子の意味は次回以降に勉強します。

[4] C 言語と同様に、英文字で始まり、英文字か下線 (_)、数字が続くような名前はもちろん、漢字やひらがな、カタカナなども使うことも可能です。

[5] これらの書式の意味や、**インタフェース**についても次回以降に勉強します。

クラス宣言の インスタンス変数、コンストラクタ、インスタンスメソッドなどの宣言 の部分には、この 3 種類以外にもいろいろな宣言[6]が現れることができます。これらの宣言を書く順番は自由[7]ですが、

1. インスタンス変数の宣言
2. コンストラクタの宣言
3. インスタンスメソッドの宣言

の順に書かれるのが普通です。

クラス宣言の例　前回までに作成したすべての Java プログラムには、必ずクラス宣言が含まれていましたが、そこには main という名前のクラスメソッドの宣言しか書かれておらず、オブジェクトの設計図としては働いていませんでした。次の 2 つのソースファイルは、クラス宣言を本来の役割であるオブジェクトの設計図として利用したプログラムの例です。

Hand.java

```
 1 import jp.ac.ryukoku.math.graphics.*;
 2
 3 /* Hand クラスのクラス宣言 */
 4 class Hand {
 5     /* このクラスのインスタンス変数の宣言 */
 6     int x, y;                        // 左端の手札の位置
 7     int deltaX = 100;                // 隣り合った手札の x 座標の差
 8     int numCards;                    // 手札の枚数
 9     Card[] cards = new Card[5];      // 手札の配列
10
11     /* このクラスのコンストラクタの宣言*/
12     Hand(int x, int y) {
13         this.x = x;
14         this.y = y;
15     }
16
17     /* このクラスのインスタンスメソッド add の宣言 */
18     void add(Card c) {
19         if (numCards < cards.length) {
20             c.moveTo(x + numCards * deltaX, y);
21             cards[numCards++] = c;
22         }
23     }
24 }
```

S0401.java

```
 1 import jp.ac.ryukoku.math.graphics.*;
 2
 3 class S0401 {
```

[6]インスタンス変数 (フィールド)、インスタンスメソッド、コンストラクタの宣言以外に、インスタンス初期化子 (コンストラクタに先だって実行されるインスタンスを初期化する手続き)、クラス変数 (静的フィールド)、クラスメソッド (静的メソッド)、クラス初期化子 (静的初期化子とも呼ばれるクラス自体を初期化するための手続き)、メンバクラス (このクラス定義のために用いられる別のクラス)、メンバインタフェース (このクラス定義のために用いられるインタフェース) の宣言が現れることができます。

[7]インスタンスやクラスの初期化を行う手続きが実行される順番には影響を与えます。

```
 4     public static void main(String[] args) {
 5         GameFrame f = new GameFrame();
 6         Deck d = new Deck();
 7         f.add(d);
 8         d.shuffle();
 9         Hand h = new Hand(160, 400); // Hand のインスタンスを生成
10         for (int i = 0; i < 5; i++) {
11             Card c = d.pickUp();
12             h.add(c);
13             c.faceUp();
14         }
15     }
16 }
```

これら 2 つのソースファイルをコンパイルすると、Hand.class と S0401.class の 2 つのクラスファイルが作られます[8]。java コマンドで実行するのは S0401 の方です。S0401 は、第 1 回に紹介した S0103 と同様の動作をします。

このプログラムでは、Hand と S0401 という 2 つのクラスが宣言されていますが、S0401 の方は、これまでと同様、main というクラスメソッドを定義するためだけに存在していて、オブジェクトの設計図としては使われていません。

一方、Hand.java に書かれている Hand クラスの宣言は、オブジェクトの設計図という本来の役割を持っていて、S0401.java の 9 行目の new Hand(160, 400) というインスタンス生成式で、この設計図に基づいてオブジェクトを生成しています。この行は Hand 型の変数 h の宣言ですが、Java では、このように、ブロックの先頭に限らず、その途中でも変数を宣言して (そのブロック内のその宣言以降で) 使うことができます。

Hand クラスは、最大 5 枚のカード (Card クラスのインスタンス) からなる 1 組の手札を表すクラスです。手札は横一列に並べて置かれます。このクラスには、次のようなコンストラクタ、インスタンス変数、インスタンスメソッドが用意されています。

Hand クラス — 最大5枚までの1組の手札

コンストラクタ	
Hand(int x, int y)	最も左の手札が (x, y) の位置となるような 1 組の手札
インスタンス変数	
int x	最も左の手札の x 座標
int y	最も左の手札の y 座標
int deltaX	隣り合った手札の x 座標の差
int numCards	手札の枚数
Card[] cards	手札の配列
インスタンスメソッド	
void add(Card c)	カード c を移動して手札に加える

4.2 インスタンス変数の宣言

Hand クラスのインスタンスは、最も左の手札の位置、隣り合った手札の x 座標の差、手札の枚数、手札となっているカード群を記憶するためのインスタンス変数を持っており、これらのインスタン

[8] 2 つのクラス宣言を、1 つのソースファイルにまとめて書いても構いません。その場合でも、コンパイルするとクラスファイルは 2 つ作成されます。

ス変数が6行目から9行目に次のように宣言されています。

```
6     int x, y;                          // 左端の手札の位置
7     int deltaX = 100;                  // 隣の手札との x 座標の差
8     int numCards;                      // 手札の枚数
9     Card[] cards = new Card[5];        // 手札の配列
```

クラス宣言の中でインスタンス変数を宣言する場合は、クラス宣言の { } のすぐ内側に、通常の変数[9]を宣言するように

型名 変数名の列 ;

の書式で宣言します。型名 の前に public や protected、private というアクセス修飾子や、final や volatile など、その他の修飾子を書くこともできます。アクセス修飾子は、プログラム中で、その変数にアクセスすることのできる範囲を指定するもので、詳しくは次回以降に解説します。また、final は、その変数が、一旦初期化されたら以降は書き換えができない (final 変数[10]である) ことを表す修飾子です。

変数名の列 の部分には、変数名 のみか、変数名 = 初期値 のように、その変数の初期値を指定したものを、, (カンマ) で区切って並べます。また、変数名 の後に [] を書いて 型名 型を要素とする配列型の変数を宣言することもできます。

インスタンス変数の変数名はクラス名と同様に自由に選べますが、Java の慣習では、numCards や cards のように、基本的に英小文字にして、2つ目以降の単語の先頭の文字を英大文字にすることになっていますので、これに従いましょう。

初期値が指定されない場合、インスタンス変数の値は、プリミティブ型なら偽 (false) あるいは 0 で、参照型なら null 参照で初期化されます。これは、配列が (初期値が指定されずに) 生成されたときの配列要素の初期化のされ方と同じです。

インスタンス変数はいくつでも宣言することができます。このクラスのインスタンスが生成される際に、そのインスタンスの各インスタンス変数を記憶するためのメモリ領域が確保され、インスタンスごとにインスタンス変数の値が記憶されます。

4.3 コンストラクタの宣言

Hand.java の 12 行目から 15 行目までは、Hand クラスのコンストラクタの宣言が書かれています。第1回に説明したように、コンストラクタは、生成されたオブジェクトの初期化を行うための手続きです。コンストラクタは

クラス名 (仮引数宣言の列) コンストラクタ本体

の形で宣言します[11]。クラス名 は、このコンストラクタ宣言を囲んでいるクラス宣言のクラス名

[9] クラス宣言の {} のすぐ内側ではなく、ブロック ({} で囲まれた文の並び) の中で宣言された変数を**局所変数 (ローカル変数)** と呼びます。S0401.java の5行目の f や9行目の h、11行目の c がそうです。

[10] 前回説明しました。

[11] クラス名 の前に public や private などのアクセス修飾子を書くこともできます。また、クラス名 の直前 (アクセス修飾子より後) に**型引数**と呼ばれるものの列が、コンストラクタ本体 の直前に **throw 節** と呼ばれる書式が来ることがあります。

と等しくなります。仮引数宣言の列 の部分は、型名 仮引数名 を , (カンマ) で区切って並べます。Hand.java の 12 行目は

```
Hand(int x, int y) { ··· }
```

のようになっていますが、x と y という仮引数がともに int 型だからといって

```
Hand(int x, y) { ··· }    // 文法エラー
```

のように書くことは許されません[12]。

コンストラクタ本体 の部分は {} で文の並びを囲んだ形[13]になります。ここに、生まれたばかりのオブジェクトが、このクラスのインスタンスとしての仕事をする準備を整える (インスタンスの初期化をする) 手続きを記述します。コンストラクタには戻り値がないことに注意してください。

Hand クラスの宣言では、ただ 1 つのコンストラクタが宣言されていますが、引数の数や引数の型で区別できるようにして、複数のコンストラクタを宣言することもできます[14]。このことをコンストラクタの**多重定義 (オーバーロード)** と呼びます。

4.4 this

コンストラクタの本体では、生成されたばかりの (コンストラクタで初期化しようとしている) インスタンスを this というキーワードで参照することができます。Hand クラスのコンストラクタでは、

```
this.x = x;
this.y = y;
```

のように、インスタンス変数の x、y に、コンストラクタの仮引数の x、y の値をそれぞれ代入しています。this.x や this.y が、オブジェクトのインスタンス変数にアクセスするときの書式である

オブジェクトを表す式 . インスタンス変数名

の形をしていることに注意してください[15]。この例では、仮引数の名前とインスタンス変数の名前が重なってしまっていますが、this. があるかないかで、そのどちらを意味しているのかを区別することができます。

他の変数と名前が重ならない場合は、インスタンス変数にアクセスする際の this. を省略することもできます。たとえば、Hand クラスのコンストラクタは、仮引数の名前を付け替えることで、

```
Hand(int xCoord, int yCoord) {
    x = xCoord;
    y = yCoord;
}
```

---

[12] この辺りは C 言語と同じです。

[13] 基本的には**ブロック**と呼ばれるものと同じ形ですが、一部、コンストラクタだけで許される特別な形の文があります。

[14] たとえば、第 1 回の「付録 : カードゲーム向けクラスライブラリ」で紹介されているように、GameFrame クラスには 2 つ、Card クラスには 3 つのコンストラクタが用意されています。

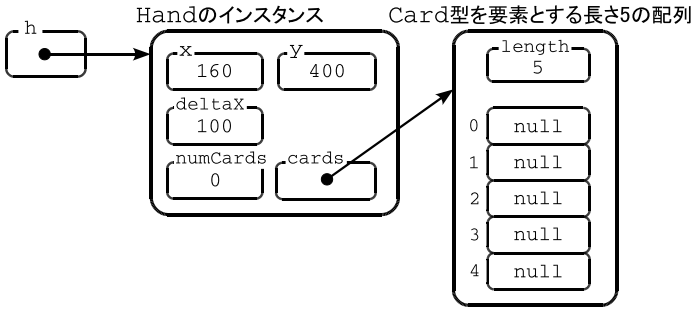
[15] 第 2 回で説明しました。

のように宣言することも可能です。代入演算子 = の左辺となっている x と y は、それぞれ this.x と this.y の省略形です。

インスタンス生成式とコンストラクタ本体の実行

S0401.java の 9 行目の new Hand(160, 400) というインスタンス生成式が評価されると、Hand クラスの新しいインスタンスが生成され、インスタンス生成式に引数として与えた 160 と 400 という int 型の値が、Hand クラスのコンストラクタの仮引数 x と y に代入されて、その本体が実行されます。コンストラクタの実行が完了すると、コンストラクタによって準備の整ったオブジェクト (Hand クラスのインスタンス) が、インスタンス生成式の値となります。

9 行目で生成された Hand クラスのインスタンスは、main メソッドで宣言された変数 h に代入されて、次の図のような状況となります。



インスタンス変数 x、y には、コンストラクタによって、それぞれ 160 と 400 が代入されています。また、deltaX には、このインスタンス変数の宣言 (7 行目) で指定された初期値 100 が代入されています。一方、numCards の宣言には初期値が指定されていませんので既定の初期値の 0 となっています。cards は

```
Card[] cards = new Card[5];
```

のように、初期値を指定して宣言されていますが、この初期値 new Card[5] の評価は、Hand クラスのインスタンスが生成される度に行われることに注意してください。この初期値を評価することで、Card 型の要素 5 個からなる配列オブジェクトが生成されますが、この配列の生成は、Hand クラスのインスタンスごとに行われますので、cards が指す配列オブジェクトは、Hand のインスタンスごとに異なります。

4.5 インスタンスメソッドの宣言

Hand.java の 18 行目からは、Hand クラスのインスタンスメソッド add の宣言が始まっています。インスタンスメソッドの宣言は

戻り値の型名 メソッド名 (仮引数宣言の列) メソッド本体

という書式で行います[16]。これは、C 言語での関数定義の書き方とほぼ同じです。戻り値のないメ

[16] 戻り値の型名 の前に (コンストラクタの宣言と同様に) public や private などのアクセス修飾子や、abstract や final、synchronized など、他の修飾子を書くこともできます。また、戻り値の型名 と メソッド名 の間にに型引数の列が、メソッド本体 の直前に throw 節が来ることがあります。

ソッドの場合は、C 言語と同様に、戻り値の型名 を void とします。メソッドの名前は、クラス名や変数名と同様に選ぶことができますが、Java の慣習では、変数名と同じく、基本的に英小文字にして、2 つ目以降の単語の先頭文字を英大文字にすることになっています。

仮引数宣言の列 の部分はコンストラクタの宣言の場合と同じ形です。また、メソッド本体 も同様で、{} で囲まれた文の並び (ブロック) の形になります[17]。

Hand クラスの宣言では、add という名前のインスタンスメソッドが次のように宣言されていますが、その本体では、引数として渡されたカード (仮引数 c が指す Card のインスタンス) を、手札を並べる所定の位置に移動した後、インスタンス変数 cards の指す配列の空き要素に記憶し、numCards の値を 1 増やしています。

```
18     void add(Card c) {
19         if (numCards < cards.length) {
20             c.moveTo(x + numCards * deltaX, y);
21             cards[numCards++] = c;
22         }
23     }
```

if 文を使って、手札がすでに 5 枚 (numCards の値が 5) に達している場合は、このメソッドは何もしないようにしていることに注意してください。

コンストラクタの宣言と同様に、インスタンスメソッドの宣言の本体でも、そのメソッドの仕事を行っているオブジェクト自身を this というキーワードでを表すことができます。19 行目から 21 行目に現れている変数名 x や y、deltaX、numCards、cards は、本来なら this.x、this.y、this.deltaX、··· のように書くべきものですが、(コンストラクタの宣言でそうであったように) 変数名が重ならない場合は this. を省略できることを利用して、単にインスタンス変数名のみを書いています。また、たとえば、this.add(···) のように、自分自身のインスタンスメソッドを呼び出す場合でも、this. を省略して、add(···) のように書くことが可能です。

Hand クラスの宣言では、ただ 1 つのインスタンスメソッドが宣言されていますが、名前で区別できるようにして、たくさんのインスタンスメソッドを宣言することができます。また、同じ名前のメソッドでも、引数の数や、引数の型で区別できる場合は、(コンストラクタの場合と同様に) 複数宣言することができます。同名のメソッドを同じクラスに複数宣言するとことを、メソッドの**多重定義 (オーバーロード)** と呼びます。

メソッド起動式とメソッド本体の実行

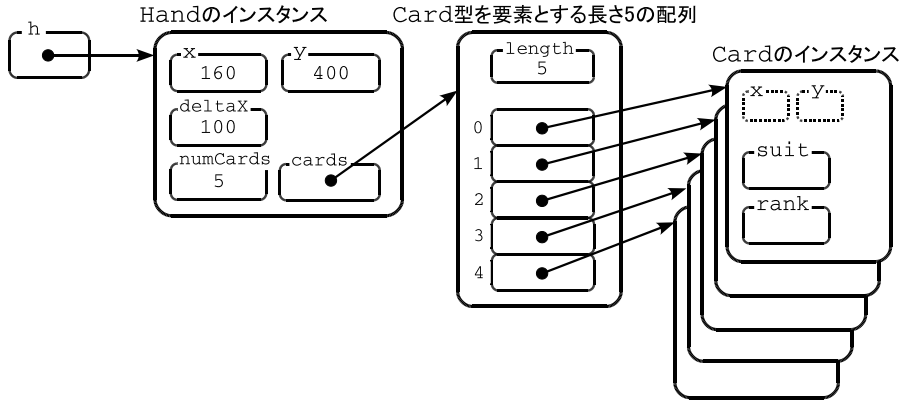
S0401.java では、main メソッドの次の for 文の中で、変数 h の指す Hand クラスのインスタンスの add メソッドを起動しています (12 行目)。

```
10         for (int i = 0; i < 5; i++) {
11             Card c = d.pickUp();
12             h.add(c);
13             c.faceUp();
14         }
```

[17] 戻り値の型名 の前に absract という修飾子を含めた場合は、この部分はブロックではなく ; (セミコロン) となります。このように宣言されたメソッドは**抽象メソッド**と呼ばれます。抽象メソッドについては次回以降に勉強します。

この時、デッキから取り出されたカードが引数として渡され、これが、Hand.java の 18 行目の仮引数 c に代入されて、add メソッドの本体 (19 行目から 21 行目) が実行されます。そこで、そのカードが移動するとともに、(h が指していたオブジェクトの) インスタンス変数 cards が指す配列の要素に記録されていきます。

この for 文が繰り返しを終えた時には、変数 h の指すオブジェクトの状況は、次の図のようになっているはずです。



4.6 演習問題

1. Hand のクラス宣言に、次のような仕事を行う 3 つのインスタンスメソッド draw、count、get の宣言を追加しなさい。

`void draw(Deck d)`	デッキ d から 1 枚カードを取って手札に加え、表向きにする。デッキが空の場合や、手札がすでに 5 枚ある場合は何もしない。
`int count()`	手札の枚数を戻り値として返す。
`Card get(int i)`	手札のうち、添字 i のカードを戻り値として返す。i が負だったり、手札の枚数以上である場合は null を返す。

ただし、draw が、デッキから取ったカードを手札に加える際には、自分自身のインスタンスメソッド add を起動するようにしなさい。

2. Hand のクラス宣言に、さらに次のようなインスタンスメソッド discard の宣言を追加しなさい。

`void discard(int i, Pile p)`	手札の添字 i のカードを山 p に (移動して) 捨てる。捨てたカードより添字の大きい (右の) 手札の添字は 1 つずつ小さくなり、1 つ左に移動する。i が負だったり、手札の枚数以上である場合は何もしない。

このように Hand クラスが変更できたら、次のプログラム S0402.java を実行してみましょう。

S0402.java

```
1 import jp.ac.ryukoku.math.graphics.*;
2
3 class S0402 {
4     public static void main(String[] args) {
```

```
 5          GameFrame f = new GameFrame();
 6          Deck d = new Deck(1);
 7          Pile p = new Pile();
 8          f.add(d);
 9          f.add(p, 100, 240);
10          d.shuffle();
11          Hand h = new Hand(160, 400);
12          do {
13              while (h.count() < 5) {
14                  h.draw(d);
15              }
16              int i = 0;
17              while (i < h.count()) {
18                  if (!h.get(i).isJoker()
19                          && h.get(i).rank != Rank.ACE) {
20                      h.discard(i, p);
21                      continue;
22                  }
23                  i++;
24              }
25          } while (h.count() < 5);
26      }
27  }
```

このプログラムは、ジョーカー 1 枚を含むデッキから 5 枚のカードを手札に加えた後、不要な手札を何枚か捨てては、デッキからカードを引いて手札を 5 枚にする、ということを繰り返し、最終的には、次の図のように、手札をエース 4 枚とジョーカーにします。

数理情報セミナー・第 4 回・終わり