計算機プログラミング I (9)

# データの抽象化

2000 年 12 月 5 日 増原 英彦

## 1 複数の値からなるデータ

### 1.1 例:複素数

前回の課題の 1 つは「$(x+iy)$ のような複素数を、$a[0] = x,\ a[1] = y$ であるような大きさ 2 の`double`型の配列 $a$ で表わす」というものであった。これに対して「何故 (2 つの変数ではなくて) 配列を使って表わす必要があるのか?」という疑問もあっただろう。何故だろうか?

その理由は複素数を**1** つのデータとして扱うためである。複素数は 2 つの実数が組みになっているが、計算式の中では**1** つの値として書かれる。これと同じく、プログラム中でも 1 つの値として扱いたいということだ。

また、複素数どうしの足し算をするメソッドが定義できれば`c = add(a,b);` のようにして足し算ができるだろう。しかし、メソッドは答えを **1** つしか返せないので、複素数は 1 つの値として扱えなければいけない。

つまり、

> 複数のデータがまとまって **1** つの値として扱われるのなら、
> プログラム中でも **1** つの値として扱えるべき

ということである。

同様のことは、「2 次元平面上の点の位置」($x, y$ 座標の組) や「日付」(年月日の組) を扱う場合にも言えるだろう。

課題では、実数型の配列を使い、0 番目を実部、1 番目を虚部と約束して表わしていた。そのため、例えば "`a[0]`" と書いあれば「`a` の実数部を使っている」と人間が読みかえてあげることになっていた。この程度の例であれば大した問題ではない。しかし、沢山のデータから成る値を 1 つの配列としてまとめてしまうと、何番目が何を表わしているのかはすぐに分からなくなってしまうだろう。つまり、

> 値をまとめるのはよいことだが、配列を使ってやるのはあまりよくない

ということにも注意しておこう。

### 1.2 例: 名簿

次に名簿管理プログラムを作ることを考えてみよう。このプログラムは、人物 1 人につき、氏名、生年月日、身長、体重などの情報を扱い、ファイルから情報を読みとる、年齢の若い順に出力する、特定の条件に一致する人達だけを表示させる、といった機能があるとする (表 1 )。

| 氏名 | 生年月日 | 身長 (m) | 体重 (kg) |
|---|---|---|---|
| 吉田　浩 | 1971-11- 5 | 1.80 | 85 |
| 平尾　博司 | 1975-12-31 | 1.74 | 68 |
| 中込　伸 | 1970- 2-16 | 1.83 | 99 |
| ⋮ | | | |

表 1: 名簿データの例

この場合も**1** 人分のデータは**1** つの値とするのがよかろう。しかし、1 人分のデータには、文字列型 (氏名)、整数 (生年月日)、実数 (身長や体重) など、異なるデータ型がある。一方、配列は中身が全て同じ型でなければいけないので、複素数に習って配列を使ってまとめることはできない。つまり、

配列では**1** つにまとめることのできない情報もある

ということだ。

## 2　オブジェクト: 複数の値をまとめた値

プログラミング言語には、様々な種類のデータを 1 つにまとめる機能があり、複合データ、構造体、レコード型、データ型などと呼ばれている。Java 言語ではオブジェクトに複数のデータを 1 つにまとめる機能がある[1]。

オブジェクトを作るためには、まずクラスを定義しなければならない。(これまで、クラスはプログラムをまとめる単位として使ってきたが、それ以外にデータをまとめる単位として使うこともできる。) 例えば、名簿の中の 1 人物のデータは以下のように定義できる:

```
class Person {// 人物データ: 名簿管理プログラムで使用
  String name;                          // 名前
  int birthYear, birthMonth, birthDate; // 生年月日
  double height, weight;                // 身長と体重
}
```

- `Person` は、定義されるクラスの名前である。この名前は、後でオブジェクトを使う際に、型の名前として使われる。
- `String name;` というのは、オブジェクトの中にしまわれる値 (インスタンス変数と呼ばれる) の型 (`String`) と名前 (`name`) を定義している。この名前は、後でオブジェクトの中から値をとり取したり、オブジェクトの中の値を変更するときに使われる。

  以下の 2 行も同様にインスタンス変数の型と名前を定義している。

次に、定義された`Person` クラスを使う例として、ファイルから 1 人分のデータを読みとって、それを画面に表示するだけのプログラムを示す (図 1 )。

[1]オブジェクトには、いろいろな機能があるのだが、ここではデータ 1 つの値にまとめる機能に注目している。

```
1  class ReadAndShow {// 人物データを読んで表示するプログラム
2
3    public static void main(String[] args) { // ここから実行される
4      BufferedReader input;
5      (略: "members"という名前のファイルから読む準備をする)
6      Person p = readPerson(input); // ファイルから人物データを読む
7      System.out.println(personToString(p)); // データを表示する
8    }
9
10   // ファイルから人物データを 1 人分読んで Person オブジェクトを作って返す
11   static Person readPerson(BufferedReader input) {
12     Person p = new Person(); // Person オブジェクトを作る
13     // 以下、人物の各情報を読んで Person オブジェクトの中にセットする
14     p.name = readLine(input);           // 名前
15     p.birthYear = readInteger(input);  // 生まれた年
16     p.birthMonth = readInteger(input); // 生まれた月
17     p.birthDate = readInteger(input);  // 生まれた日
18     p.height = readDouble(input);      // 身長
19     p.weight = readDouble(input);      // 体重
20     return p;    // 完成したオブジェクトを返す
21   }
22
23   static String personToString(Person p) {  // 人物データを文字列に変換する
24     return ("名前: " + p.name + ", 誕生日:" +
25             p.birthYear + "-" + p.birthMonth + "-" + p.birthDate +
26             ", 身長:" + p.height + ", 体重:" + p.weight);
27   }
28
29   // ファイルから 1 行読んで文字列・整数・実行として返す
30   static String readLine(BufferedReader input) {(略)}
31   static int readInteger(BufferedReader input) {(略)}
32   static double readDouble(BufferedReader input) {(略)}
33 }
```

図 1: オブジェクトを使う例 (完全版は web ページにある)

- 12 行目のPerson p = new Person(); の左辺は、Person 型の変数p を宣言している。このように、クラスの名前はそのクラスのオブジェクトの型になる。
- 12 行目の右辺はPerson クラスのオブジェクトを 1 つ作っている。作られたものはインスタンスあるいはオブジェクトと呼ばれる。

  ここで作られたオブジェクトの中身、つまり、インスタンス変数の値は空である。整数や実数の場合はゼロに、文字列などの場合は空であることを示すnull という値が入っている。
- 14 行目の p.familyName = readLine(input); の左辺のように

  オブジェクトを表わす式.インスタンス変数名

  と書くと、オブジェクトのインスタンス変数に値を代入することができる。
- 同様に、24 行目の... + p.name + ... のように、オブジェクトのインスタンス変数から値をとり出すこともできる。
- 11 行目の static Person readPerson(...) で定義しているようにメソッドの返り値の型にクラス名を書き、20 行目のreturn p; のようにオブジェクトを返すことができる。

  この返り値は 6 行目の Person p = readPerson(input); のように受け取って変数にしまうことができる。
- 7 行目の showPerson(p); のように、メソッドの引数としてオブジェクトを渡すこともできる。

  23 行目のstatic String personToString(Person p) のようにして、メソッドの定義側でオブジェクトを受け取ることができる。

練習 **9-1:** (名簿)　ファイルから人物データを読みとり表示する名簿プログラムを作れ。ファイルの先頭には人数が書いてあり、名簿はPerson クラスのオブジェクトの配列で表現するものとする。

練習 **9-2:** (複素数)　複素数を表わすクラスを定義せよ。それを用いて、課題 2-2 のメソッドを定義し直せ。

練習 **9-3:** (日付・時刻)　日付 (年月日) と時刻 (時分秒) を 1 つにつまとめた値をを表わすクラスを定義せよ。日付時刻を表示するメソッド、2 つの日付時刻 $t1, t2$ を受け取って前後関係を判定し、$t1$ の方が昔であればtrue、そうでなければfalse を返すメソッドを定義せよ。

練習 **9-4:** (並べかえ)　名簿データを体重順に並かえ、重い方から 3 人だけ表示するプログラムを作れ。(並べかえのアルゴリズムは付録参照)

練習 **9-5:** (肥満度)　体重を身長の 2 乗で割ったものを肥満度と定義する。

1. クラスPerson の定義に肥満度を表わすインスタンス変数を加えよ
2. データをファイルから読みとる際に、自動的に肥満度を計算してセットするようにreadPerson を変更せよ

3. 表示の際にも肥満度を表示するようにpersonToStringを変更せよ

4. 名簿を肥満度順に並べかえ大きい順に表示させよ

# 3 オブジェクトが置かれる場所

すでに見たように、オブジェクトを、変数に代入したり、メソッドの引き数や返り値として受け渡しすることができる。実はそのときに渡されているのは、オブジェクトのある場所であって、オブジェクトそのものではない。

このことを理解するために、メソッドの引数として整数を渡す場合を復習しよう。これまで (つまり整数などの場合)、「使う側と使われる側で別々の箱が用意される」と説明してきた。例えば、下のようなプログラムではメソッドmainの変数xには10という値がしまわれており (2行目)、sum(x)が実行されるときに、変数xから値がとり出され (3行目)、メソッドsumFrom1Toの変数nにその値がしまわれる (6行目)。従って、メソッドsumFrom1Toの中で変数nを減らしていっても (7行目)、mainのxには影響しない。

```
1  public static void main(String[] args) { // ここから実行される
2    int x = 10;                    // 変数 x に 10 を入れる
3    int sigmaX = sumFrom1To(x);    // メソッドを使って計算
4    System.out.println("1 から"+ x + "までの和は" + sigmaX + "です。");
5  }
6  static void sumFrom1To(int n) { // 1 から n までの和を求める
7    for (int sum = 0; 0 < n; n = n - 1) // n を 1 ずつ減らしながら
8        sum = sum + n;                  // sum に加えてゆき、
9    return sum;                         // 最終的な sum を返す
10 }
```

オブジェクトを受け渡す場合には事情が変わったように見える。下のプログラムでは、人物データを読みとって (9行目) 表示し (10行目)、次にメソッドchangeUintsを使って (11行目) 単位系を変えてまた表示する (5行目)。さらに、mainに戻ってからもう一度表示している (12行目)。

```
1  static void changeUnits(Person p) {// 身長・体重をフィート・ポンドに直す
2    p.height = p.height/0.3048;   // 1 フィート=30.48cm
3    p.weight = p.weight/0.45359237;// 1 ポンド=453.59237g
4    System.out.println("changeUints:"+ personToString(p)); // 表示
5  }
6  public static void main(String[] args) {
7    ...(略)...
8    Person p = readPerson(input);                  // 人データを読んで
9    System.out.println("main(1):    "+personToString(p)); // まず表示
10   changeUnits(p);                                  // 単位系を変えて
11   System.out.println("main(2):    "+personToString(p)); // また表示
12 }
```

これを実行すると、下のように表示される。

```
main(1):    名前: 吉田浩，誕生日:1971-11-5，身長:1.8，体重:85.0
changeUint:名前: 吉田浩，誕生日:1971-11-5，身長:5.905511811023622，体重:187.3929
main(2):    名前: 吉田浩，誕生日:1971-11-5，身長:5.905511811023622，体重:187.3929
```

つまり、メソッドの定義側でオブジェクトの中身を書き換えた場合、メソッドを使う側でもその変化が見える、ということである。このことは、次のようにして理解するとよい:

- `new Person()` のような式を実行すると、(どこかに) 新しいオブジェクトが 1 個できる。
- 変数にオブジェクトを代入する (例えば `p = new Person();`) と、変数p にはオブジェクトの参照(reference) が入る。
- オジェクトのインスタンス変数を変更したり読みとったりする操作は、オブジェクトの参照を通して行う。つまり、`p.height=1.69;` とすると、p が指し示している場所にあるオブジェクトの`height` というインスタンス変数を書き換える。
- メソッド呼出で渡されるのはオブジェクトの参照である。従って、メソッドの使用側と定義側では、渡された参照を通して同じオブジェクトを操作していることになる。

この性質は一見混乱するが、人物 1 人につき 1 つのオブジェクトを割り当てるという考え方からは自然なことである。

そのため、成績管理プログラムを作っていて、例えば「出席日数が足りない人の成績を半減する」という処理をする際に、

- 名簿の中から出席日数不足の人だけからなる作業用の名簿を作る
- 作業用の名簿中の全員の成績をそれぞれ半分にする

ということをするだけで、もとの名簿から見た場合でも成績が変更されていることになる。もし、メソッドを使ってデータを受け渡す際に、データのコピーが作られるとしたら、どれが正しいデータだかを把握するため、かえって混乱してしまうだろう。

練習 **9-6:** 作業用の名簿

1. 名簿 (人物データの配列) を受け取り、体重が $x$kg 以上の人だけを抜き出した新しい名簿を作り、それを返すメソッドを定義せよ。

   ただし、あらかじめ人数が何人いるか分からないので、与えられた名簿と同じ大きさの配列を作り、余ったところにはには`null` という値を入れておくことにする。(この`null` は、オブジェクト配列などの値がないことを示すために用意された特別な値である。)

2. 与えられた名簿の全員の名前の前に “*” という文字を追加するメソッドを作れ。

   ただし、値がない場合もあるので、`if (members[index] != null) { ... }` のようにして、「値があれば何かをする」という書き方にするとよい。

3. 上のメソッドを組み合わせて、全員の情報を表示する際に体重が 80kg 以上の人を “*” 印付きで表示させてみよ。

練習 9-7: メソッドの引数と参照　下のプログラムを実行した場合、どのように表示されるか。また、その理由を説明せよ。

```
1  static void changeUnits(Person p) {// 身長・体重をフィート・ポンドに直す(?)
2    Person pCopy = new Person(); // 新しく人物データを作って中身をコピー
3    pCopy.name = p.name; pCopy.birthYear = p.birthYear;
4    pCopy.birthMonth = p.birthMonth; pCopy.birthDate = p.birthDate;
5    pCopy.height = p.height/0.3048;   // 1フィート=30.48cm
6    pCopy.weight = p.weight/0.45359237;// 1ポンド=453.59237g
7    p = pCopy; // pに pCopy を代入
8    System.out.println("changeUnits:"+ personToString(p));
9  }
10 public static void main(String[] args) {
11   ...(略)...
12   Person p = readPerson(input);          // 人データを読んで
13   System.out.println("main(1):    "+personToString(p)); // まず表示
14   changeUnits(p);                        // 単位系を変えて表示
15   System.out.println("main(2):    "+personToString(p)); // また表示
16 }
```

# 4　オブジェクトの中のオブジェクト

いままでは、オブジェクトの内に整数・実数・文字列などのデータが入っている例だけを見ていたが、オブジェクトの中にオブジェクトや配列を入れることも可能である。

これを使うと、データをもっと整理して扱うこともできる。例えば、人物のデータであれば、

- 名前は姓と名に分かれている
- 生年月日は日付というオブジェクトにまとめた方がよい

といったことを、オブジェクトの中にオブジェクトを入れることで実現できる。

プログラムとしては、必要な数だけのクラスを定義し、インスタンス変数の型にクラス名を書くようにすればよいので以下のようになる:

```
1  class Person {                    // 人物データのためのクラス
2    Name name;                      // 氏名
3    Date birthday;                  // 誕生日
4    double height, weight;          // 身長・体重
5  }
6  class Name {                      // 氏名のためのクラス
7    String familyName, givenName; // 姓・名
8  }
9  class Date {                      // 日付のためのクラス
10   int year, month, day;           // 年・月・日
11 }
```

このように定義しておくと、

- 人物データに「データを入力した日」情報を加える場合、Date クラスのインスタンス変数をもう 1 つ加えるだけでよくなる。
- Date クラスに「曜日」情報を加える場合、Person クラスの定義を変える必要がない

など、プログラムを作ってゆく際の労力が減る。

練習 **9-8:** (人物データの細分化)　上のような方針で人物データのクラス定義を細分化せよ。(ただし、名前に関しては、ファイルに書かれたデータが姓名に分かれていないので、そのままでもよい。) その際、練習 9-2 で作った日付クラスを使うようにしてみよ。表示の際に、練習 9-2 で作ったメソッドを使うようにせよ。

# 付録: 並べかえ

大きさ $n$ の整数の配列に入っているデータを、大きい順に並べかえるには、

1. 0 ~ $(n-1)$ 番目の中で最大のものを探し ($m$ 番目にあったとする)、0 番目と $m$ 番目を入れ替える、
2. 1 ~ $(n-1)$ 番目の中で最大の (つまり全体で 2 番目に大きい) ものを探し ($m$ 番目にあったとする)、1 番目と $m$ 番目を入れ替える、
3. 2 ~ $(n-1)$ 番目の中で最大の (つまり全体で 3 番目に大きい) ものを探し ($m$ 番目にあったとする)、2 番目と $m$ 番目を入れ替える、

   …… 

のようにすればよい[2]ので、例えば下のようになる。

```
 1  static void sort(int[] data) { // 配列 data の中身を大きい順に並べかえる
 2    // 0 番から順に大きい順に値を決めてゆく
 3    for (int current = 0; current < data.length; current=current+1) {
 4      // この時点で current より前の値は全て current 以降より大きい
 5      // current 以降の中での最大値を見つけ、data[current] と交換する
 6      int maxIndex = current;  // 最大値がある場所
 7      for (int compare = current; compare < data.length; compare=compare+1)
 8        if (data[maxIndex] < data[compare])   //もし compare 番目が大きかったら
 9          maxIndex = compare;                 //最大値の添字を変更
10      // current 以降では data[maxIndex] が最大値
11      int temporary = data[maxIndex]; // data[current] と
12      data[maxIndex] = data[current]; // data[maxIndex] を入れ替える
13      data[current] = temporary;
14      // この時点で data[current] は current より後ろの全てより大きい
15    }
16  }
```

[2]もっと頭の良いやり方がいろいろあるが、ここでは説明を簡単にするために単純かつ頭の悪いものを紹介した。