4 配列～複数の変数をまとめます～

4.1配列ってなんだ？

第 3 章では整数型や実数型などの変数型を勉強しました．一方プログラミングをしていると，同じ様なデータをいくつも使わなくてはいけないことがあります．

例えば，毎日の電話料金を記録していくとしましょう．このとき，一日ごとに別の変数に通話料金を記録したいですよね．でも・・・

```
int day1
int day2;
・・・
```

と 31 日分書くのはちょっと大変です．毎日データを入れるのも大変だし，最終的な通話料金を計算したいときも，

```
int totalPrice = day1+day2+・・・+day31;
```

としなければいけません．もう，面倒くさいことこの上ないですね．

また，画面に毎日の金額を出力しようとしたら，

```
System.out.println(day1);
System.out.println(day2);
・・・
System.out.println(day31);
```

そんなときに使えるのが，配列です．配列を使うと同じようなデータがあればひとまとめにくくって使うことが出来るのです．

配列は，変数の一種ですが，同じ型の変数を指定数だけ確保できるというのが最大のポイントです。たとえば、整数型 int を 100 個確保することもできるし、実数型 double を 20 個確保しておくことも出来ます。もちろん、確保した個数だけ自由自在に使うことが出来ます。ようするに、今までは変数ひとつにデータひとつしか格納できなかったのですが、今後は配列を使うことで変数ひとつにいくつでもデータを保存できるようになるのです。31 日分のデータを一つの変数で表せることが出来たら便利ですよね．

さあ、それでは、配列を使ったプログラムを見てみましょう。まずは、配列の作り方から学びます。ここでは，先ほどの 31 日分のデータを日にちごとの通話料金を保存する配列として作ってみましょう．

```
int[] callCharge = new int[31];
```

この配列は，全部で３１個のデータを保存することが出来ます．

また，31 個のデータはそれぞれこんな名前で利用することが出来ます．

```
days[0] = 100;
days[1] = 200;
・・・
System.out.println(days[2]);
```

これで初日の通話料は 100 円で，二日目は 200 円，最終日は 250 円ということが表現できるようになりました．

ちなみに，

```
int[] days;
```

の様に配列として宣言された変数を「配列変数」と呼び，

```
days[0];
```

のようにその中の一つに着目したものを，「配列の要素」といいその時の[]内の数字（ここでは 0）を「添え字」と呼びます．

```
days[2];
```

と書いた物は，「**配列** days の 2 番目の**添え字**によって得られる**要素**」と呼ぶことになります．ちょっと特殊な用語ですが，よく使われるので覚えておいてください．

では，詳しく配列に説明していきましょう．

4.2配列の作り方

配列の作成方法には二つの方法があります．一つは，値を最初から指定して作成する方法，もう一つは空の配列を作る方法です．

4.2.1　値指定の配列

まずは，最初から値が入っている配列の作り方を学んでみましょうか．このような配列を作るには，こんな書き方をします．

```
int[] dailyPriceArray = {100,200,150,100,130,150, 0};
```

３１日分のデータを書くのはちょっと面倒なので一週間分，つまり 7 日分のデータだけを持っている配列を作る例です．

このように，配列を宣言するには配列に入る型名のあとに[]を書けば終了です．また，その配列に入るデータは{}の中に,区切りで記述します．まとめると，

```
型名[] 配列名 =  {値 0, 値 1, 値 2…};
```

こんな感じになります．

先ほどの例だと int 型なので値も整数ですが，整数以外にも実数や文字列でも作ることが出来ます．

```
double[] taxedPriceArray = {105, 210, 157.5, 136.5, 157.5, 0};
String[] weekArray = {"月", "火", "水", "木", "金", "土", "日"};
```

ちなみに，各値を利用したいとき配列名の後ろに[]とその中に添え字をつけて，こんな感じで行います．

```
int sum = dailyPriceArray[0]+dailyPriceArray[1];
taxedPriceArray[1] = 315;
System.out.println(weekArray[4]);
```

このように記述した場合，普通の変数と全く同じように扱うことが出来るようになります．

4.2.2 大きさ指定の配列作成

さて、先の例ではいったん配列を作ると同時に、入れるデータを決めていました。しかし、実際こう書くのは結構面倒くさいですよね。配列を宣言した場合、あらかじめ配列の中身を決めておかなければいけないわけですから、例えば一年 365 日分のデータが入った配列を作ろうと思ったら、中身を 365 個決めておかなければいけないわけです。それはちょっと面倒です。365 個くらいならまだ頑張れるかもしれませんが、もし携帯電話会社の顧客が 1 月 1 日に使った電話料金データを全部配列で扱いたいと思ったら・・・100 万，200 万のデータは普通に使わなければ行けなくなりそうです．…もう気が遠くなりそうです。

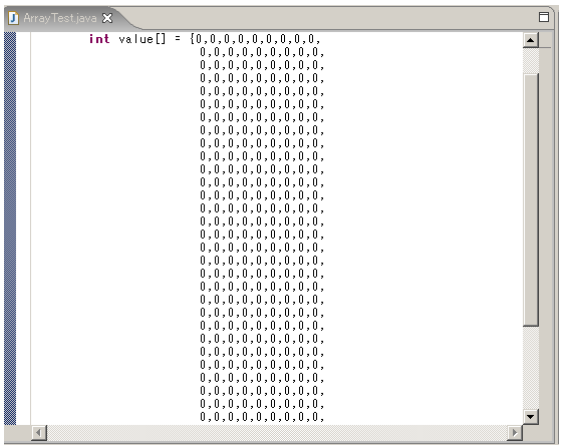

図 1　気が遠くなりそうな作業

しかも、最初に入れるべき値が決まっているならかまいませんが、後々入れるべき値を決めたい！という場合には非常に面倒ですよね。

そんなときは、こんな方法で配列を作成すると便利です。

```
int[] yearPriceArray = new int[365];
```

この例では、365 個の整数型の配列を確保しています。こう書いておけば、一年分の通話料を全部記録しておくことが出来ますよね。一年の最初にこの配列を確保しておいて，具体的にいくら使ったかは毎日更新していけばいいんですから．もし一つ目の方法で配列を作成しようとしたら、一年分の支出がすでに分かっていないとできませんからね。

ここでポイントとなるのが、

```
int[] yearPriceArray = new int[365];
```

この部分です。new というのはおまじないだと思っておいていただいて、int[365]の部分を見てみましょう。

まず、int は整数型の配列を作るよ、という意味です。すでに配列名のまえに変数型を書いていたのに、また変数型を書くなんて二度手間のような気がしますが、これはおまじないのようなものだと思ってください。そういうルールなのです。ちなみに，なぜこんな書き方をするかはクラスの継承を勉強すると分かるようになってきます．が，それはまだまだ先の話ですので，今は聞き流しておいてください．

new int のあとの[]の中には、この配列にいくつ値を格納するかを指定します。この場合は、365 個のデータを入れることができる配列を作ることになります。ま，間違えようがない話ですね．

ところで、ここでは 365 日分のデータを保存できるようにこうしていますが、考えてみると世の中にはうるう年というものがあります。この場合、366 日分のデータを保存しなければいけません。そんなときは、

```
int[] yearPriceArray = new int[366];
```

とすれば Ok です。いや、楽チンですね。

同じように、実数や文字列の配列を作る場合はこんな感じ。

```
double[] yearTaxedPriceArray = new double[365];
String[] monthArray = new String[12];
```

4.3配列の利用

4.3.1 配列の要素を取得

配列はいくつかの変数=要素を一まとめにする変数ですが、一まとめにした変数一つ一つにアクセスできなければ意味がありません。

そこで、配列の中のそれぞれの要素にアクセスしてみましょう。せっかくですので、先ほど作った曜日を記述した配列の要素にアクセスしてみましょう。各要素にアクセスするためには，配列名[添え字]という書き方をします．

```
String[] weekArray = {“月, “火, “水, “木, “金, “土, “日”};
```

```
String date = weekArray[1];
System,.out.println(date);
```

さて、どう表示されるでしょうか？1 つめの要素ですから、「月」と表示されそうですよね。

じゃあ、見てみましょう。

```
火
```

あれ、予想外に二番目に入れたはずの「火」が出てきました。

じつは、配列の[]の中(添え字)は 0 からスタートするのです。いや、分かりづらいという気持ちはわかりますが、そういうものなのです。0 からスタートして、配列に確保した数-1 までの値を入れることができます。この場合ですと、曜日が 7 日分ありますので、7-1=6 までの添え字を入れることができます。

この配列の場合，添え字と要素との関係は以下のようになります。

添え字	0	1	2	3	4	5	6
要素	月	火	水	木	金	土	日

というわけで，「月」と書いてある要素を取り出すには，

```
String[] weekArray = {“月, “火, “水, “木, “金, “土, “日”};
String date = weekArray[0];
System,.out.println(date);
```

と書く必要があります.

```
月
```

最初のうちは多少混乱するかもしれませんが、そういうものだと覚えてしまいましょう。そのうち 0 から始まらないと気が済まなくなってきてしまいます．筆者など，「一年の月が 0 月から始まってくれれば便利なのに・・・」とまで思っちゃったりしてます.

ちなみに、C や C++などのプログラム言語でも配列の添え字は 0 からスタートします。Java は C の文法を継承していますので、配列の添え字も同様に 0 からスタートすることになっているんですね。

4.3.2 配列要素の操作

配列の要素を取り出して表示させるのに成功しましたが、次は配列の要素に対して行うことができる色々な操作をみていきましょう。基本的に，配列は要素一つ一つが変数とまったく同じ働きをしますので，配列に対して行うことが

そのために、まず中身を決めていない配列を作成して、任意の位置に値を入れることを目指してみます。

```
int[] ary = new int[7];
ary [0] = 100; //代入
```

```
ary [1] = 50+31; //足し算の代入
int x =9;
ary [2] = x; //変数の代入
ary [3] = 0;
ary [3]++; //インクリメント
ary [4] = ary [0]+ ary [1]; //要素同士の加算と代入
int j = 5;
ary [j] = j*2; //添え字を変数で指定して代入
for(int i= 0; i < ary.length; i++){
        System.out.println("ary["+i+"]="+ary[i]);
}
```

ここでは、長さが 7 の配列を作成して、色々な方法で要素を代入しています。

なお，最後の三行にちょっと見慣れない物が書いてありますが，ここは配列の要素全てを表示するおまじないだと思ってください.

では、その結果がどうなったのか見てみましょう。

```
ary[0]=100
ary[1]=81
ary[2]=9
ary[3]=1
ary[4]=181
ary[5]=10
ary[6]=0
```

というわけで、配列の要素に対して色々操作してみました。

配列は，変数の集合ですから，添え字を使って要素を指定すれば，その要素はまるでそこに変数がひとつあるだけかのように扱うことができます。++などの操作も使えるあたりから、それがわかるのではないでしょうか。

なお、最後添え字が 6 の配列データに対しては何の操作もしていません。このような場合、0 が値として入っています。これについては後で説明しますね．とりあえず，整数や実数の変数型を配列にした場合，初期値は 0 になると覚えておいてください.

4.3.3 配列の長さを知る

さて、配列のデータを利用する場合、0 からデータ数-1 までの添え字が使えます。では、添え字がデータ数異常だった場合はどうなってしまうのでしょうか？使えるといわれている範囲を超えているので、当然使えないことは予想できるのですが、好奇心旺盛なわれわれは試してみずにはいられません。やっちゃだめといわれることほどやりたくなるのが人情です。

```
String[] weekArray = {"月", "火", "水", "木", "金", "土", "日"};
System.out.println(weekArray[10]);
```

さあ、こんな悪いプログラムを書いてしまいましたよ。果たしてないはずのデータをよこせ、という振り込め詐欺みたいな要求に Java はなんとこたえてくるのでしょうか？

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
```

おっと、エラーが出てプログラムが終了してしまいました。つまり、データがないところのデータを取ろうとしてはいけないよ、ということです。

ちなみに、この ArrayIndexOutOfBounds というエラーの意味は「配列の添え字が範囲を超えています」というそのままの意味です。

このように配列が確保していない部分まで使おうとすると、プログラムが停止してしまいます。そこで、配列の長さ以上に使おうとしないように、あらかじめ配列の長さを知りたいですよね。

そこで、配列の長さを知る方法を学んでおきましょう。一週間の曜日名を書いた配列を一つ作成して，その長さを知るプログラムが以下の物です.

```
String[] weekArray = {"月", "火", "水", "木", "金", "土", "日"};
int len = weekArray.length;
System.out.println("配列の長さは"+len);
```

ここでは，int 型の変数 len に配列 weekArray の長さを記録しています．このプログラムの実行結果は,

```
配列の長さは 7
```

というわけで，ちゃんと一週間の長さを取得することが出来ました.

さて，ここで注意ですが、配列の添え字は **0～配列の長さ-1** まで許されていますから、配列の長さを添え字として使うことは出来ません。Array.length は、最初に確保したデータ数と同じ値を返しますから、ここでは 10 になるわけです。array[10]が範囲外となるように、

```
array[array.length]
```

とはできないわけです。もし、配列の一番最後の値を使いたいのであれば、

```
array[array.length-1]
```

と書かなければいけません。これは、Java を勉強して 1 年以内の人が最もやりがちなミスのひとつといわれています。皆さんはそんな普通の初心者がやりがちなミスを回避するように、気をつけましょう。

では，最後に配列の長さを調べるプログラムの構文をご紹介しましょう.

```
配列変数名.length
```

これで終わりです．簡単ですね.

ところで、先ほどのソースを見てこう書けばいいんじゃないかと思った方はいません

か？

```
String[] weekArray = {"月", "火", "水", "木", "金", "土", "日"};
int lengthWeekArray = 7;
```

するどいですね。このように書くことで，あらかじめ配列の長さを記録することが出来て，わざわざ weekArray.length などと書く必要がなくなるでしょう．

しかしながら，もしもある日突然一週間が 8 日になってしまったら・どうなるでしょうか？

```
String[] weekArray = {"月", "火", "水", "木", "金", "土", "日","天"}; //天王星が曜日の仲間入り！
int lengthWeekArray = 8;
```

この場合，なんと二箇所もプログラムを修正しなければいけなくなります．いや，確かに一週間が 8 日になるなんて事はなかなかありえないことだとは思いますが，惑星の数が一つ減ってしまうような時代です．いつ何がおきるか分かりませんよ．

そんとき，修正する場所が一箇所でよければ自信を持ってそこだけ修正をすればよいのですが，二箇所となるともしかするとどちらかを修正し忘れてしまうかもしれません．

もちろん，こんな単純な例だと間違えようがありませんかもしれませんが，これからどんどん複雑なプログラムを書くようになっていくと，修正部分が増えれば増えるほど修正し忘れる可能性が高くなっていきます．

さらに，もしこれが 2 箇所じゃなくてもっと多かったらどうでしょうか？10 箇所直さなきゃいけないとか、場合によってはどこで配列の長さを使うか忘れてしまう場合だってあります。配列の長さを間違えたばっかりに、正しく値が 0 にならずに、あとでトンでもない計算結果が出てきてしまうかもしれません。

また、さらに恐ろしいことに、もし、一週間が 8 日になるならばよいですが，6 日になってしまったら，どうでしょうか？もしこのとき修正し忘れると・・・

```
String[] weekArray = {"火", "水", "木", "金", "土", "日"}; //皆が嫌いな月曜日を抹殺
int lengthWeekArray = 7;
```

この場合、曜日の長さが 7 日あると勘違いしてプログラムを書いてしまうと，ArrayIndexOutOfBoundsException が発生することになるでしょう。その結果プログラムが途中で止まってしまいます。

一方で、array.length を使えば修正もしなくて良いうえに、エラーが発生する心配もぐっと少なくなります．確実に配列の長さ以上の添え字を使わずにすむわけですから。

というわけで、配列の長さを必要とするときは無理せずに length を使うようにしましょう。

4.4 多重配列

4.4.1 多重配列の基本

さて、配列はいくつかのデータをまとめて確保する方法だという話をしていましたが、実は配列を使うと「配列変数を配列を使ってまとめる」なんてことが出来てしまいます。

一年分の通話料を記録しようと思ったときに,

```
int[] callPriceArray = new int[365];
```

とするのも悪くありません.

が，Java では多重配列というものが許されていますので,

```
int[][]callPriceArray = new int[12][31];
```

という書き方が出来てしまいます．なにやら今度は[]が二つずつ出てきていますね。このような配列を多重配列といいます。

ここでは、31 日分の支出データを 12 か月分用意しています。つまり、12×31 のデータを作成しています。

これによって「31 個の要素を持った配列を 12 個持っている配列を作成」しているのです。つまり，配列を入れる配列を作っているわけですね.

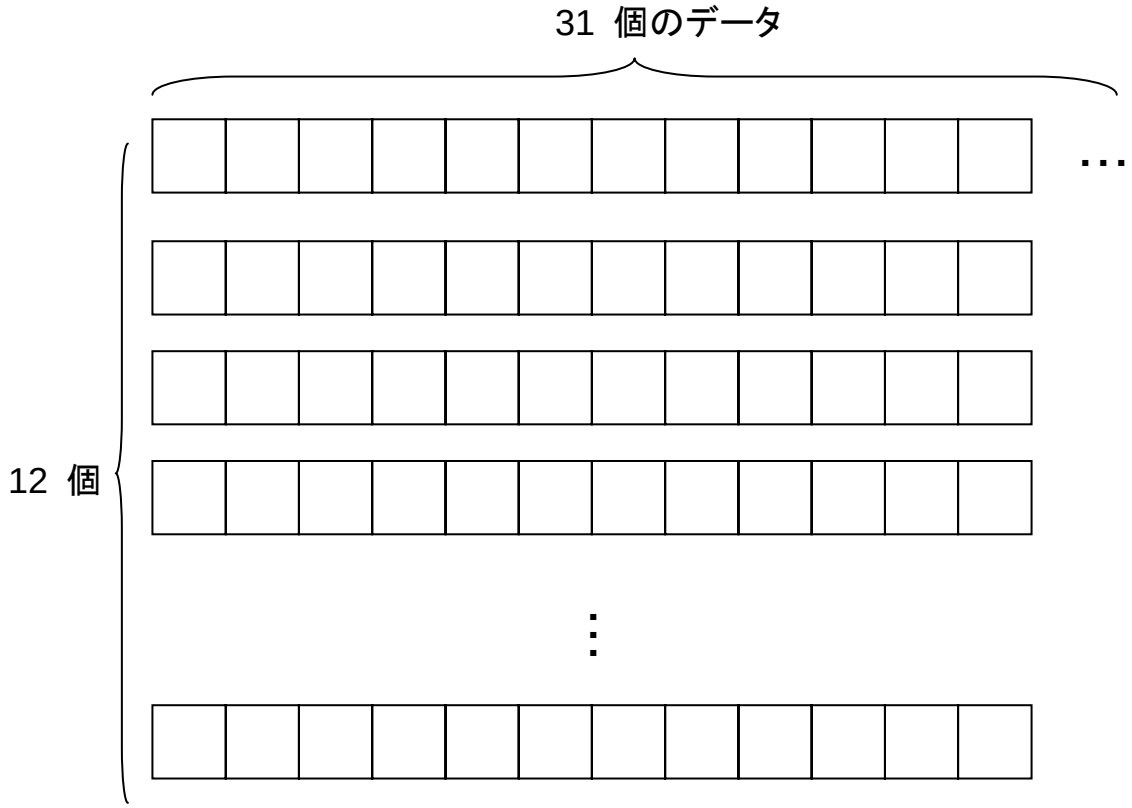

図 要素数が 31 個の配列を 12 個用意

理系の方ですと、

- 普通の配列はベクトル
- 二次元配列は行列

とイメージするとわかりやすいかもしれません。

このようにして作成した配列は，二つの[]に添え字を入れることで通常の配列と同じようにアクセスしたり値を入れ替えたりすることができます。

例えば，1 月 1 日に 500 円，5 月 9 日に 10 円分通話したとすると，

```
callPriceArray[0][0] = 500;
callPriceArray[4][8] = 10;
```

と書くことで記録することが出来るようになります．これならぱっと見ていつの記録なのかが分かって便利ですよね．添え字は 0 から始まることにさえ注意すれば，何月何日のデータなのか一目で分かるわけです．

4.4.2 長さの異なる多重配列

ところで、ここでは毎月 31 日分の支出データを作成するようにしています。が、実際には 2 月は 28 日までしかないし、4 月も 30 日までしかありません。このままでは、調子に乗って 2 月 31 日の支出まで書いてしまうことになりかねません。後日なぞの支出を確認して頭を悩ませることになってはつまらないので、2 月の配列には 28 日までしかないような配列を変更しようと思います。

各月の日数を指定して多重行列を作るには以下のようにします。

```
int[][]callPriceArray = new int[12][]; //①

callPriceArray[0] = new int[31]; //②
callPriceArray[1] = new int[28];
callPriceArray[2] = new int[31];
callPriceArray[3] = new int[30];
callPriceArray[4] = new int[31];
callPriceArray[5] = new int[30];
callPriceArray[6] = new int[31];
callPriceArray[7] = new int[31];
callPriceArray[8] = new int[30];
callPriceArray[9] = new int[31];
callPriceArray[10] = new int[30];
callPriceArray[11] = new int[31];
```

ここでは、まずは①で配列を要素とする配列を 12 個作っています．

そして，②でその配列にその月の日数分の要素を格納できる配列を作成しているのです。

ところで、ここで作った多重配列で、2 月 31 日に 1 万円も通話料を使ったという架空請求をしてみたらどうなるでしょうか？添え字は利用したい日付-1 で指定することを忘れないでくださいね.

```
callPriceArray[1][30] = 10000;
```

さあ、実行してみましょう。何が起こるか、皆さん予想してみてくださいね。

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 30
```

さあ、予想は当たりましたか？

2 月のデータには 28 日までしかデータを格納していませんので、31 日目のデータにアクセスしようとすると、配列の外にはみ出したデータにアクセスすることになりますので、例のごとく ArrayIndexOutOfBoundsException が発生してしまいました。

4.4.3 さらに多重配列

ちなみに，多重配列というくらいですから，3 重や 4 重にすることだって出来てしまいます.

```
int[][][] callPriceInLife =  new int[100][12][31];
```

こんな感じで, 100 歳まで生きたときに毎年何月何日にいくらずつ使ったのかを全部記録するだって出来ちゃいます．これで，遠い子孫にまで電話代を節約した人生を送ったことを知らせることができますね.

もちろん，利用する場合はこんな感じになります.

```
callPriceInLife[19][11][23] = 0;
```

というわけで，二十歳のクリスマスイブに誰とも電話していないこともバレバレです.

4.5間違えやすい配列

4.5.1 配列の初期値

配列を作ったとき，その初期値はどうなっているでしょうか？

```
String[] addressArray = {"警察:110", "消防:119", "時報:117"};
int[] dailyPriceArray = {100,200,150,100,130,150, 0};
```

このような書き方をしたときは，指定された値が入ることが分かります.

一方,

```
int[] dailyPriceArray = new int[365];
String[] addressArray = new String[10];
```

と書いた場合，その要素にはどんな値が入っているのでしょうか？ちょっと表示してみましょうか.

```
int[] intArray = new int[10];
```

```
double[] doubleArray = new double[10];
boolean[] booleanArray = new boolean[10];
String[] stringArray = new String[10];



System.out.println(intArray[0]);
System.out.println(doubleArray[0]);
System.out.println(booleanArray[0]);
System.out.println(stringArray[0]);
```

これで，int 型，double 型，boolean 型，String 型の配列を作成し，最初の要素を出力しています．ただし，その前にそれぞれの要素の値は代入していませんので，初期値がそのまま出力されるはずですよね．さあ，その結果は・・・？

```
int 配列の初期値は 0
double 配列の初期値は 0.0
boolean 配列の初期値は false
String 配列の初期値は null
```

という結果が出ました．

みると分かるとおり，int 型の場合 0，double 型の場合 0.0 となり，数字系の配列は初期値として 0 が入ることが分かります．

一方，boolean 型の場合，0 はありませんよね．true/false のどちらかしか入らないのですから．で，どちらが初期値になるのかといえば，false の方なんですね．これにはあまり深い意味はないと思いますが，とにかく boolean 型の配列の場合，全要素の値は false になっていることを覚えておきましょう．

そして，最後の String 型の初期値です．なんとなく，空白文字が入っていそうな気がしていましたが，驚いたことに「null」という文字が初期値として設定されているように見えます．

いったいこの「null」とは何者なのでしょうか？実は，これは「値がありません」ということを意味しているのです．

つまり，String 型配列の要素の初期値として”null”という文字列が入っているわけではなく中身は「空っぽ」なのです．

この「空っぽ」状態については，クラスについての説明の際に詳しくお話しますので，今は String 型の配列は，初期値は「空っぽ」だという風に覚えておいてください．

最後に，それぞれの変数型の配列の初期値を示したいと思います．

変数	タイプ	初期値
int 型	整数	0
short 型	整数	0

long 型	整数	0
double 型	実数	0.0
float 型	実数	0.0
boolean 型	論理値	false
char 型	文字	0
byte 型	バイト	0
String 型	文字列	null

表 配列にしたときの初期値

4.5.2 配列の添え字は 0 からスタート．最大でも length-1 まで

何度もしつこいようですが，配列でもっとも間違えやすいのは添え字の問題です．なれないうちは，添え字が 0 からスタートすることをついつい忘れてしまいます．

```
String[] addressArray = {"警察:110", "消防:119", "時報:117"};
String firstAddress = addressArray[1];
System.out.println("最初のアドレスは"+ firstAddress);
```

しかし，これでは，

```
最初のアドレスは消防：119
```

となってしまい，警察に電話を掛けたかったのに消防に電話をかけてしまうことになります．

最初の要素をとりたい場合はこうします．

```
String firstAddress = addressArray[0];
System.out.println("最初のアドレスは"+ firstAddress);
```

これで，

```
最初のアドレスは警察：110
```

と，ちゃんと警察の番号を取得できました．

一方もし，最後の要素を取得したい場合はこうします．

```
String lastAddress = addressArray[addressArray.length-1];
System.out.println("最後のアドレスは"+lastAddress);
```

これで，

```
最後のアドレスは時報：117
```

となります．

これはもう決まり文句のように覚えておいてください．

また，配列の大きさと同じ添え字で要素を取得しようとしてしまいがちです．

```
System.out.println("3 つ目のアドレスは"+addressArray[3]);
```

なんてやると，

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
```

ほら怒られてしまいます．必ず何番目の要素を取得したいのかを考えたら，その番号-1 の添え字を指定しなければいけません．

このようなミスは単純に書いている場合はあまり多くないかもしれませんが，配列の要素を利用する場合，その添え字が int 型の変数になっていることがよくあります．

```
int idx = 0;
String idxAddress = addressArray[idx];
System.out.println(idx+"のアドレスは"+idxAddress);
```

このような場合，idx の値が 0～addressArray.length-1 までという保証はなくなります．もしこのアドレス帳プログラムを誰か別の人が利用しようとしたら idx の範囲が 0～addressArray.length-1 というルールをよく知らず，

```
int idx = 10;
```

などとやってしまうかもしれませんからね．

このようなミスをなくすためにはどうすればよいでしょうか？

実は，次章でならう if 文を使うと簡単に誤りを回避することが可能です．

```
if(idx >= 0 && idx < addressArray.length){
        String idxAddress = addressArray[idx]; //①
        System.out.println(idx+"のアドレスは"+idxAddress);
}
```

このように記述することで，idx が配列の添え字として取りうる範囲にある場合のみ①以下の処理を行うようになります．

詳しくはすぐに if 文のところで説明しますが，こうすることで配列の添え字違反が発生しづらくなりますので，このテクニックを覚えておいて下さい．

4.5.3 配列のコピー

配列を使っていると、ときどき配列の中身を全部コピーしてしまいたいときがあります。そんなときどうすればよいでしょうか？

ここでは，携帯電話のアドレス帳をバックアップするシステムを開発しているとしましょう．間違ってアドレス帳を消してしまったときにバックアップがあれば安心ですからね．

さあ，レッツバックアップ．

```
String[] addressArray = {"警察:110", "消防:119", "時報:117"};
```

```
String[] backupAddressArray = addressArray;
```

これで，間違えてアドレスを消してしまっても，backupAddress から復旧すればいつでもアドレスを取り戻すことが出来ますよね．

```
addressArray[0] = "天気予報:177";
addressArray[1] = "電話の新設・移転・各種ご相談:116";
addressArray[2] = "海上の事件・事故の急報:118";
```

と，ここで間違って全アドレスを消して，別のデータを上書きしてしまいました！大変．火事の現場に居合わせても消防署に知らせることも出来ません．

でも，大丈夫．バックアップがありますからね．

早速バックアップデータを復旧しましょう．

```
addressArray = backupAddressArray;
System.out.println(addressArray[0]);
System.out.println(addressArray[1]);
System.out.println(addressArray[2]);
```

これで，バックアップデータを復旧できたはずです．

```
天気予報：177
電話の新設・移転・各種ご相談：116
海上の事件・事故の急報：118
```

ななな、なんとバックアップデータを復旧させたはずなのに，上書きしたデータ残ってしまっています．なぜこんなことが・・・！？

原因を探るべく，backupAddressArray の中身を確認してみましょう．

```
System.out.println(backupAddressArray[0]);
System.out.println(backupAddressArray[1]);
System.out.println(backupAddressArray[2]);
```

```
天気予報：177
電話の新設・移転・各種ご相談：116
海上の事件・事故の急報：118
```

なんと，バックアップデータも一緒に上書きされているじゃありませんか．コピー先を変更したらコピー元の値まで変更されてしまっているわけです。よく、双子の兄が怪我をすると弟の体にも同じ傷ができる、とかいう都市伝説もありますが、それと同じような恐怖体験を配列でもできてしまうのです。まあ、こわい。

って、そんなわけはありません。実は、配列を= で代入すると値をコピーするわけではなく、配列に別名をつけるだけという動作を行ってしまいます。そのため、一見コピーして二つの配列ができたような気がするこの操作も、実際は二つの変数は同じ配列を意味し

ていることになります。いうなれば、芸名をつけたようなものです。

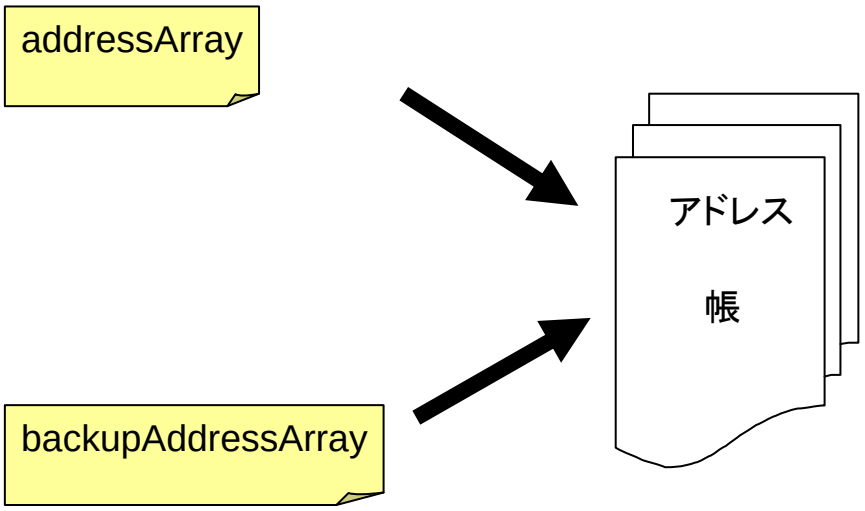

図 1 名前は違っても同じものを指しています

では、同じ値を持った二つの配列を作成するためにはどうすればよいでしょうか？
ひとつ考えられる方法としては、こんなやり方があるでしょう。

```
String[] addressArray = {"警察:110", "消防:119", "時報:117"};
String[] backupAddressArray = new String[3];

backupAddressArray[0] = addressArray[0];
backupAddressArray[1] = addressArray[1];
backupAddressArray[2] = addressArray[2];
```

これで配列は二つになって、片方の値を変更してももう片方も変更されてしまう、ということはありません。

```
addressArray[0] = "天気予報:177";
addressArray[1] = "電話の新設・移転・各種ご相談:116";
addressArray[2] = "海上の事件・事故の急報:118";
```

間違えて上書きしても・・・

```
addressArray[0] = backupAddressArray[0];
addressArray[1] = backupAddressArray[1];
addressArray[2] = backupAddressArray[2];

System.out.println(addressArray[0]);
System.out.println(addressArray[1]);
System.out.println(addressArray[2]);
```

こうすることによって復旧可能です.

```
警察：110
消防：119
時報：117
```

ほら,復旧できました.

しかし、ちょっと面倒くさいですね。次章で勉強する for 文を使えばもう少し簡単に掛けますが，いずれにせよ大変なのは間違いありません.

というわけで、Java では配列をコピーするための方法が用意されています。その方法を利用すれば、一行で配列をコピーできるので楽チンです。それが，この System.arraycopy という命令です.

```
String[] addressArray = {"警察:110", "消防:119", "時報:117"};
String[] backupAddressArray = new String[3];
System.arraycopy(addressArray, 0, backupAddressArray, 0, addressArray.length);
```

この例だと，addressArray の中身をそっくりそのまま backupAddressArray にコピーしています.

```
addressArray[0] = "天気予報:177";
addressArray[1] = "電話の新設・移転・各種ご相談:116";
addressArray[2] = "海上の事件・事故の急報:118";

System.arraycopy(backupAddressArray, 0, addressArray, 0, backupAddressArray.length);
System.out.println(addressArray[0]);
System.out.println(addressArray[1]);
System.out.println(addressArray[2]);
```

間違えて上書きした場合も同様に，今度は backupAddressArray から addressArray へコピーを行います.

その結果,

```
警察：110
消防：119
時報：117
```

ちゃんとバックアップしておいたアドレスが復旧しました．良かったよかった.

というわけで，配列をコピーしたい場合はこの System.arraycopy を利用しましょう.

その書式は以下のとおりです.

```
System.arraycopy(コピー元配列, コピー開始位置, コピー先配列, コピー位置, コピーする長さ);
```

まず，コピー元の配列を指定します.

次に、どこからコピーを始めるかを書きます。通常は最初から全てコピーすることが多

いので，0 を指定することが多いのですが，2 番目の要素からコピーしたい場合などは，ここに 1 と書きます{footnote:2 じゃないですよ．1 ですよ}.

次に、コピー先の配列を書きます。このとき，コピー先の配列にはコピーするだけの大きさが備わっていなければいけません．つまり，3 個の要素をコピーしたいのなら大きさ 3 の配列が必要です.

また、コピー先の配列のどこにコピーするかを指定します。そのまま配列のコピーを作成したい場合は 0 ですし、すでにある配列の途中から入れたい場合は、コピー先の添え字を書きます。

最後に、どのくらいの長さをコピーするかを指定します。配列の全データをコピーしたければ、配列の長さを書き込みます．普通はコピー元配列の length を使って指定することになるでしょう．アドレスのバックアップの例だと，そうしていましたよね.

これでコピーは終了です.

配列をどうしてもコピーしなければいけない場合は，必ずこのように System.arraycopy を使いましょう.

ちなみに，System.arraycopy を使う場合，以下の点に注意してください.

- コピー先は，コピー元と同じ型の配列を用意する
- コピー元はコピーする要素を格納できるだけの大きさを確保する

これらを守らないとコピーに失敗することになりますのでご注意を.

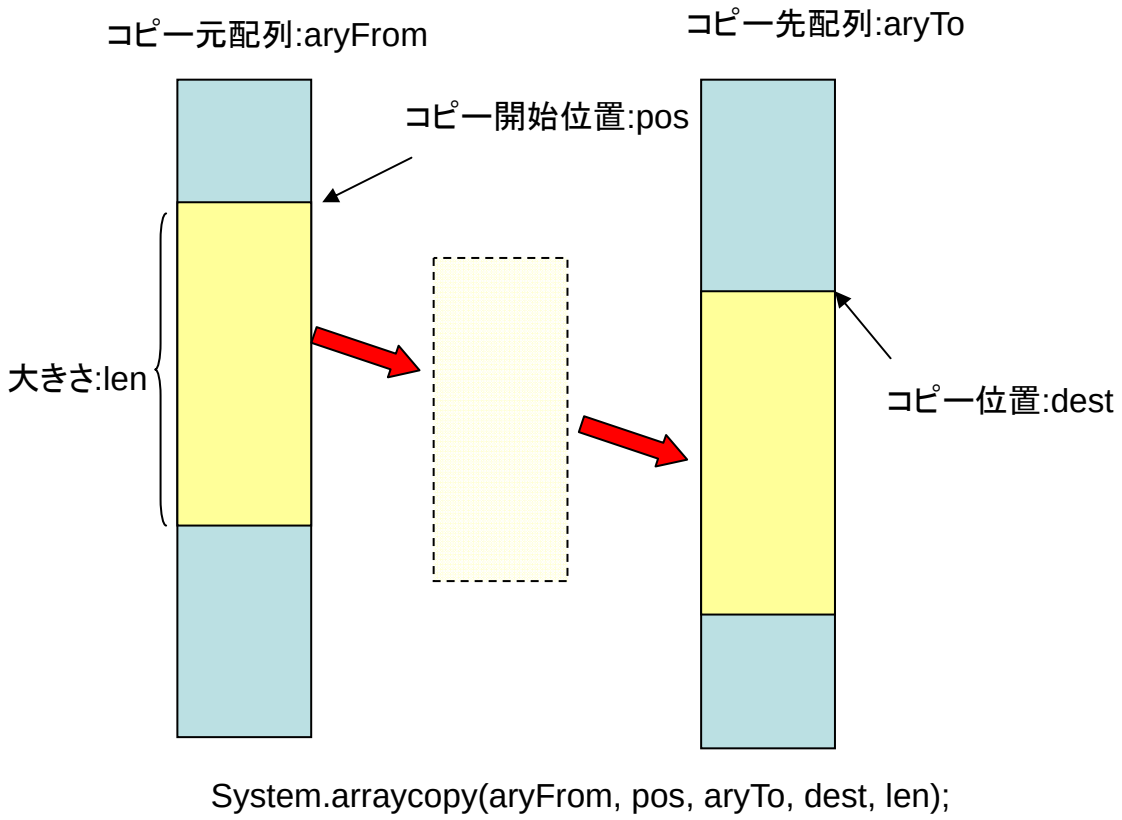

図 System.arraycopy はこう使う！

4.5.4 配列は大きさを変えられない

アドレス帳などを作っているときに，最初に 3 人分のアドレスが入った配列を作ったとします．さて，この次に新しい友人から電話番号を聞いた場合，どうすればよいでしょうか？

```
String[] addressArray = {"警察:110", "消防:119", "時報:117"};
addressArray[3] = "友人:○×○×";
```

わーい，と思って早速まだ使われていない 4 番目のデータとして添え字 3 を使って要素を追加しようとすると・・・

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
```

ショック．配列の長さが足りなくてデータを追加できません．ということは，せっかくもらった友人の電話番号を記録することが出来ないじゃありませんか．

というわけで，addressArray の配列の大きさを変更しようともくろんでみましょう．そのままでは変更できませんから，こんな風にしてみたらどうでしょうか？

```
addressArray = new String[4];
addressArray[3] = "友人:○×○×";
```

お，なんかこれならうまくいきそうですよね．

addressArray を大きさ 4 の配列に変更して，その最後に友人のアドレスを入れているのですから．事実，このプログラムを実行してもエラーは発生しません．やった！

・・・と思うのはちょっと時期尚早です．

こうやって作ったアドレス帳の全データを見てみましょう．

```
System.out.println(addressArray[0]);
System.out.println(addressArray[1]);
System.out.println(addressArray[2]);
System.out.println(addressArray[3]);
```

ちゃんと全員分入っているでしょうか？わくわく．

```
null
null
null
友人：〇×〇×
```

なんと初めから入っていた警察署などのアドレスが消えて初期値である null になってしまいました．友人のアドレスを入れたかったばっかりにこれまで記録しておいたアドレスが消えてしまったのです．これはショック．

というわけで，このアイデアも失敗です．

この結構いけそうだったアイデア，なぜ失敗したのかというと，実は

```
addressArray = new String[4];
```

この部分は，「新しい配列を作るよ」という意味なんですね．つまり，addressArray にはまったく新しく作った要素数が 4 つまで入る String 型の配列が入ってしまっていたのです．それじゃあ，これまでのアドレスは消えてしまうわけです．

では，どうすればよいのでしょうか？

次のような方法がもっとも簡単な配列の長さの変更方法です．

```
String[] backupAddressArray = addressArray; //①
addressArray = new String[4]; // ②
System.arraycopy(backupAddressArray, 0, addressArray, 0, backupAddressArray.length);//③
```

①では，配列のコピーのときはやってはいけなかったはずの配列の代入を行っています．これは何をやっているのかといえば，addressArray そのものに別名をつけているんですね．図 1 のように，addressArray が意味する配列に backupAddressArray という別名をつけました．これによって，警察の番号などが入ったアドレス帳の名前が backupAddressArray となったのです．なぜこんなことをしたのかは，すぐに分かります．

次に，②で addressArray として新しい要素が 4 つ入る配列を作成します．このとき，下からあった警察の番号などが入ったアドレスは addressArray という名前ではなくなってしまいます．もし，放っておくと二度とこのアドレス帳を使うことは出来なくなってしま

います．が，①で backupAddressArray という別名をつけておきましたので，そちらの名前を使って古いアドレス帳も利用することができるのです．ちょっとテクニカルな話ですが，図をみていただければ分かりやすいかもしれません．

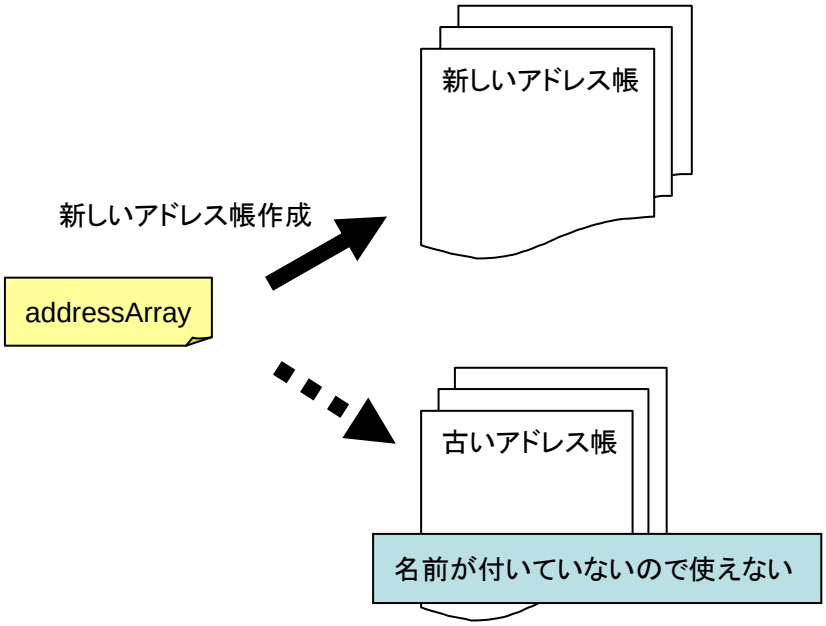

図 古いアドレス帳が使えなくなる

このようにして，古いアドレス帳と新しいアドレス帳を同時に使えるようにした上で，System.arraycopy を使って古いアドレス帳のデータを新しいアドレス帳に移動させました．

最後に，友人のアドレスを追加してみましょう．

```
addressArray[3] = "友人：○×○×";
```

これで実行してもとりあえずエラーは発生しません．しかし，エラーが出ないからと言って，先ほどの例もありますから，ちゃんとうまく前のデータも残っているかどうか確認しなければ行けませんよね．

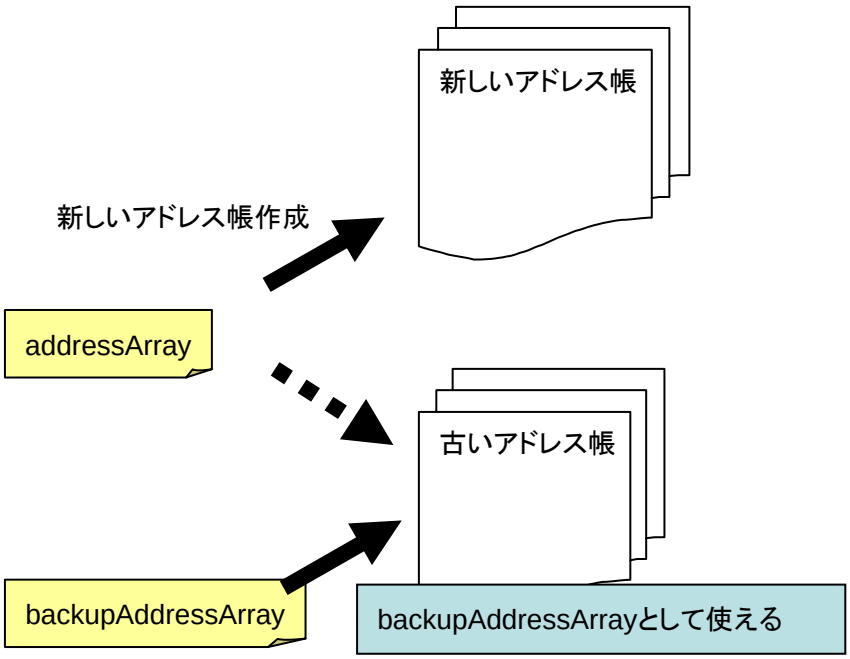

図 別名をつけて消えてなくなるのを防ぐ

```
String[] backupAddressArray = addressArray;
addressArray = new String[4];
System.arraycopy(backupAddressArray, 0, addressArray, 0, backupAddressArray.length);
addressArray[3] = "友人:○×○×";

System.out.println(addressArray[0]);
System.out.println(addressArray[1]);
System.out.println(addressArray[2]);
System.out.println(addressArray[3]);
```

さて結果はどうなったでしょうか.

```
警察：110
消防：119
時報：117
友人：○×○×
```

お，無事友人のアドレス追加に成功しました.

このようにして，配列の大きさを変更するには一端新しい配列を作ってからコピーをするという作業が必要になります.

配列の大きさを変更する構文は以下のとおりです.

```
変数型[] backupArray = 配列名;
配列名 = new 変数型[新しい要素数];
System.arraycopy(backupArray, 0, 配列名, 0, backupArray.length);
```

なお，この方法だと配列の大きさを大きくすることは出来ますが，小さくすることは出来ません．もし，配列を小さくしようと思ったら，

```
変数型[] backupArray = 配列名;
配列名 = new 変数型[新しい要素数];
System.arraycopy(backupArray, 0, 配列名, 0,新しい要素数);
```

としてください．

なお，この操作によって配列の大きさを変更することは出来ますが，System.arraycopy はあまり速度が速くないという欠点があります．つまり，配列の大きさを頻繁に変更するとプログラムの実行速度が遅くなってしまう可能性があるのです．

「配列の大きさが自由に変えられたら良いのに・・・」そんな思いが出てくるかもしれません．

実は，大きさが変えられる(というか，好きな数だけデータを追加することが出来る)配列のようなものが Java には存在します．

「だったらそれを先に教えろよ！」と思われる気持ちも分かりますが，その配列のようなものを使うには，もう少し Java について色々学んでからの方が，何かと都合がよいのです．

というわけで，もうしばらくこの大きさの変えられない配列にお付き合い下さい．

次章，制御構文で配列は大活躍する予定です．