

第4章

配列

第1章では、さまざまな数や文字列を入れることができる変数のしくみを学びました。

もちろん、作成するプログラムが大きくなると、たくさんの変数を使うことになります。

たとえば、複数の変数をすべて足したり、平均を求めたりするようなこともあるでしょう。

この章で学ぶ「配列」は、変数をより便利に使うためのしくみです。配列を使うことで、一度に多くの変数を処理することができます。

CONTENTS

- 4.1 配列のメリット
- 4.2 配列の書き方
- 4.3 配列と例外
- 4.4 配列のデータをまとめて扱う
- 4.5 配列の舞台裏
- 4.6 配列の後片付け
- 4.7 多次元の配列
- 4.8 第4章のまとめ
- 4.9 練習問題
- 4.10 練習問題の解答

4.1 配列のメリット

4.1.1 変数が持つ不便さ



湊くん、何を作っているのかな？

第1章で学んだ変数を使って、テストの点数の合計や平均を算出するプログラムを作っています。我ながらよくできていると思います。



よくできているね。でも、配列を使うとよりよくなるよ。

え～！せっかく作ったのに…。



配列をマスターしたら、プログラムが楽になるよ。

便利だから、しっかり身に付けよう。

第1章では数字や文字列などのデータを格納して扱う変数のしくみを学びました。そして第3章では、変数を用いることで分岐や繰り返しを実現できることかわかりましたね。このように、変数だけでもプログラムを書くことはできます。しかし、それだけでは少し不便なこともあります。湊くんが作成したテストの点数を管理するプログラムを見て考えてみましょう。

リスト 4-1 点数管理プログラム

```
1 public class Main {  
2     public static void main(String[] args) {
```

Main.java

```

3   int sansu = 20;      算数は 20 点
4   int kokugo = 30;     国語は 30 点
5   int rika = 40;       理科は 40 点
6   int eigo = 50;       英語は 50 点
7   int syakai = 80;     社会は 80 点
8
9   int sum = sansu + kokugo + rika + syakai + eigo;
10
11  int avg = sum / 5;   合計の算出
12  System.out.println("合計点:" + sum);
13  System.out.println("平均点:" + avg);
14 }
15 }
```

一見、問題なさそうに見えますが、このコードには不便なことがあります。

■科目が増えるたびに、それをコードに追加しなければならない

新しい科目を加えるには、taiiku(体育)などの変数を宣言した後に、合計と平均を算出している行を修正する必要があります。これを忘れるとなしい目の点数が追加されなくなってしまいます。

■まとめて処理できない

「点数の良い科目から順に並び替える」などの処理を行う場合、コードが長く、複雑になってしまいます。

これらの原因は、コンピュータが5つの科目の変数を「個々の独立したデータ」として扱っていることにあります。

私たち人間は、この5つの変数は「各科目のテストの点数を格納している変数で、**1組のもの**として計算(合計の算出など)することがある」と無意識に考えています。しかし、コンピュータにとって変数は「何の関係もないバラバラの5つの箱」でしかなく、1組のものとして扱うことができないのです。

複数のデータを1組のものとして扱うことは、私たちの身の回りに多くあります。

例として携帯電話のアドレス帳を考えてみましょう。アドレス帳に鈴木さんのデータがあるとした場合、そこには鈴木さんの「名前」「電話番号」「メールアドレス」など複数の情報を「鈴木さんの情報」として1組で管理しています。そして、アドレス帳には鈴木さんのほかにも多くの友人・知人のデータが格納されており、それらは1人1組ごとにコピーや削除が可能です。

このように、現実世界では1つのグループに属するデータをまとめて扱うことがあります。それにもかかわらず、これをプログラムで実現できないとなると、前ページに挙げたような不便が発生してしまいます。

そこで、ほとんどのプログラミング言語では、「いくつかの関係あるデータをグループにして、まとめて1つの変数に入れる」しくみが用意されています。1つの変数に複数のデータを入れるといっても、雑多に放り込むわけではなく、きちんとした構造に整理して、後から特定の値を取り出せるように格納することができます。

この構造のことを**データ構造**(data structure)といい、その代表的なものが本章で学ぶ**「配列」**です。



データ構造は、複数のデータをひとまとめにする方法で、何種類があるということですね。そして、その1つが「配列」ということですか？

そのとおりだよ。配列以外にも「マップ」や「スタック」などのデータ構造もあるんだ。



4.1.2 配列とは

配列(array)とは、同一種類の複数データを並び順で格納するデータ構造です。右の概念図を見てください。

変数のような箱が連続して並んでいます。そして、この箱の1つ

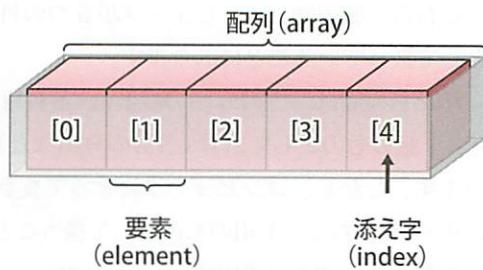


図 4-1 配列の構造

ひとつを要素 (element) といいます。要素は変数と基本的に同じで、型を持ち、データを格納できます。また、配列に含まれる各要素の型は揃える必要があります。ある要素の型が int 型ならば、その他の要素の型も int 型でなければいけません。よって配列の各要素には同一種のデータしか格納できないということになります。「要素の 0 番には数字、要素の 1 番には文字列」などの異なるデータ型は格納できません。

これにより、配列ではそれぞれの要素に値を代入して、使用することができますし、すべての要素のデータをひとまとめに扱うこともできます。そのため、「配列のすべての要素を合計せよ」、「配列のすべての要素の中身を大きい順に並び替えよ」といったことが、変数よりも簡単に行うことができます。

図 4-1 にあるように配列内の各要素には 0 番、1 番、2 番という番号が付いています。この要素の番号を添え字 (index) といい、0 から始まる決まりになっています (1 から始まるのではないことに注意してください)。たとえば、要素が 5 つあったら 0 番の要素、1 番の要素…4 番の要素と表現されます。このとき 5 番という添え字の要素はありません。



配列の要素は 0 から始まる！

最初の要素が 0 番であることは、初学者はもちろん、熟練者でもうっかり間違えることがあります。しっかり覚えておきましょう。



配列を使うには、どうすればいいんですか？

変数と同じで、使う前に準備が必要なんだ。ただし、変数より少しだけ複雑な準備が必要だから、次節でしっかり紹介しよう。



4.2 配列の書き方

4.2.1 配列の作成

配列を作成するには、以下の 2 ステップが必要です。

Step1 配列変数の宣言

Step2 要素の作成と代入

まず、Step1 で配列変数を作成します。この変数は今までの変数と異なり、代入できるのは値ではなく要素です。そして Step2 で要素を作成し、それを Step1 で作成した配列変数に代入することで配列が完成します。

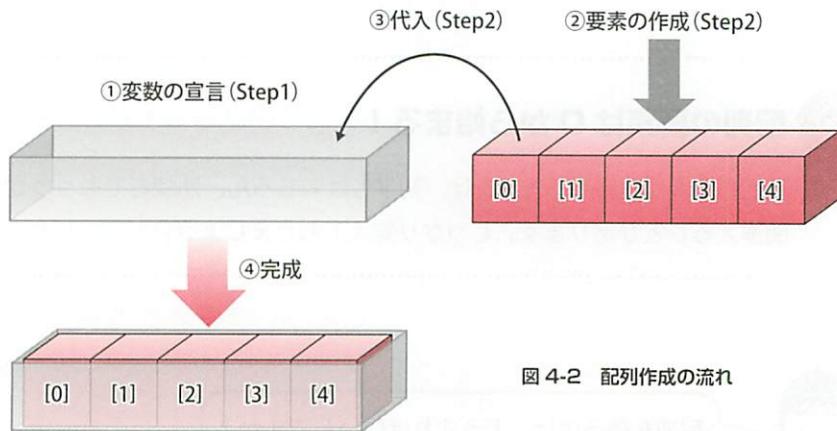


図 4-2 配列作成の流れ

Step1 配列変数の宣言

配列変数を作成するには、今まで同様に宣言が必要です。型を指定するには、要素の型の後に `[]` を付けて記述します。たとえば、int 型の要素を代入する配列の場合、次のように宣言します。

```
int[] score;
```

この宣言で使われている「int[]」型は、見た目は int 型と似ていますがまったく異なる型なので注意してください。詳しくは後で解説しますので、ここでは「int 型の要素を代入できる型」と理解してください。



代入する要素が double 型の場合は「double[]」で、String のときは「String[]」なんですね。



配列変数の作成(宣言)

要素の型 [] 配列変数名

Step2 要素の作成と代入

次に要素を作成して Step1 で宣言した配列変数に代入します。たとえば「int 型の要素を 5 個作り、配列変数 score に代入する」には、次のようにします。

```
score = new int[5];
```

new は「new 演算子」と呼ばれるもので、指定された型の要素を [] 内の数値の分だけ作成します。作成された要素は、「=(代入演算子)」で配列変数に代入することができます。



うーん。配列は変数と違って、手順がややこしいなあ。

そうよね。ここまで的内容をまとめてみましょうか。



リスト 4-2 配列の作成手順

```

1 public class Main {
2     public static void main(String[] args) {
3         int[] score;           // ①int[] score
4         score = new int[5];    // ②new int[5]
5     }
6 }
```

int型の要素を代入できる配列変数
scoreを用意 ([]が必要!)

int型の要素を5つ作成してscore
に代入し、配列scoreの完成

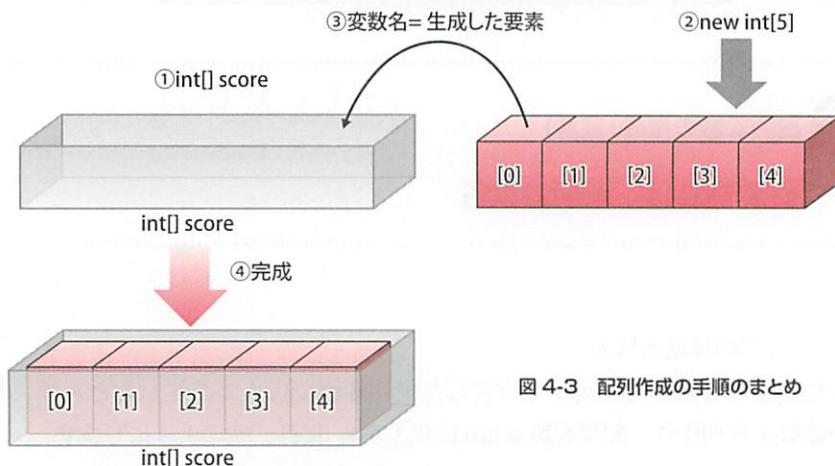


図 4-3 配列作成の手順のまとめ

また、次のようにして先ほどの「Step1 の配列変数の宣言」と「Step2 の要素の作成と代入」を同時にを行うことができます。

リスト 4-3 配列の作成手順 (Step1 と Step2 を同時に行う)

```

1 public class Main {
2     public static void main(String[] args) {
3         int[] score = new int[5];
4     }
5 }
```

Main.java

なお、いくつ要素を作ったか（配列の要素数はいくつか）は自分で覚えておかなくて大丈夫です。次のようにして「配列名.length」で調べることができます。

リスト 4-4 配列の長さを調べる

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int[] score = new int[5];  
4         int kobakos = score.length; 5 になる  
5         System.out.println("要素の数: " + kobakos);  
6     }  
7 }
```

Main.java

4
章

点数管理のプログラムで、平均を算出するときに
使うといいわよ。



なるほど！ そうすれば科目の数が増えても、平均の算
出の部分は修正しなくてもいいね。



配列の要素数の取得

配列変数名.length

4.2.2 配列の利用方法

配列に含まれるそれぞれの要素は変数と同じように扱えますが、どの要素に値を出し入れするかを指定するため、「score[1] = 10;」のように添え字を用いて利用します。ここで「**配列の最初の要素は 0 番である**」というルールを思い出してください。「score[1] = 10;」は、配列 score の**先頭ではなく、先頭から 2 番目の要素**に 10 を代入していることになります。

次のリストを見てください。これは score の 2 番目の要素に 30 を代入し、それを 6 行目で表示しています。

リスト 4-5 配列の要素に値を代入

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int[] score;  
4         score = new int[5];  
5         score[1] = 30; // 要素 score[1] に代入  
6         System.out.println(score[1]); // 要素 score[1] の中身を表示  
7     }  
8 }
```

Main.java

4.2.3 配列の初期化

ところで、変数の値を取り出す前には、必ず値を代入して初期化しなければなりません。初期化をしていない変数を利用するとコンパイルエラーが発生します。

リスト 4-6 初期化されていない変数を利用

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int x;  
4         System.out.println(x); // x が初期化されていないので、  
5     } // コンパイルエラーになる  
6 }
```

Main.java

しかし、配列の要素は自動的に初期化されます。たとえば、次のように int 型の要素を持つ配列を用意した場合、5 つの要素はすべて 0 で初期化されます。

リスト 4-7 配列の初期化

```
1 public class Main {
2     public static void main(String[] args) {
3         int[] score = new int[5];
4         System.out.println(score[0]);
5     }
6 }
```

Main.java

0が
出力される
(エラーにならない)

4
章

要素がどのような値で初期化されるかは、要素の型によって決められています。

int や double 等の数値の型	0
boolean 型	false

なお、String 型の要素は後ほど学習する「null」という値で初期化されます。

4.2.4 省略記法

これまで見てきた配列の作成と初期値の代入を同時に行うことができます。



配列作成と初期化の省略記法

- ① 要素の型 [] 配列変数名 = new 要素の型 [] { 初期値 1, 初期値 2, 初期値 3, … };
- ② 要素の型 [] 配列変数名 = { 初期値 1, 初期値 2, 初期値 3, … };

省略記法の具体例を次に示します。

```
int[] score1 = new int[] { 20, 30, 40, 50, 80 }; ) 省略記法①
int[] score2 = { 20, 30, 40, 50, 80 }; ) 省略記法②
```

4.3

配列と例外

4.3.1

範囲外要素の利用による例外の発生



先輩、コンパイルできて実行もできたんですけど、英文が表示されて止まっちゃいました。これって何ですか？

これは例外が発生したんだね。配列の使い方を間違うと表示されるエラーメッセージだよ。



リスト 4-8 点数管理プログラム（配列版）

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int[] score = { 20, 30, 40, 50, 80 };  
4         int sum = score[1] + score[2] + score[3] + score[4] + score[5];  
5         int avg = sum / score.length;  
6         System.out.println("合計点：" + sum);  
7         System.out.println("平均点：" + avg);  
8     }  
9 }
```

Main.java

合計の算出

平均の算出

合計と平均の表示

実行結果（エラー）

```
java.lang.ArrayIndexOutOfBoundsException: 5 Main.java(4)
```

:

:

湊くんが、どこで間違えたかわかりましたか？配列 score の要素数は 5 つなので要素の添え字は [0] ~ [4] ですね。しかし、このコードは 4 行目で score[5] を使ってしまっています。

このように、存在しない要素をコード内で使っていてもコンパイルは成功します。しかし、プログラムを動かすと、その行を処理しようとした際に **ArrayIndexOutOfBoundsException** というエラーメッセージが表示されプログラムが中断してしまいます。このようなエラーを、特に例外(exception:エクセプション)といいます。

例外については、詳しくは第 15 章で解説します。現時点では、実行中に「～Exception」と表示されて中断したら、例外というエラーが発生したと判断できるようにしておきましょう。



あれほど、先頭の要素の添え字は [0] だと注意されていたのに…。

これは経験者でも、うっかりやってしまうミスだよ。

ArrayIndexOutOfBoundsException が発生したら、「エラーの原因は存在しない要素を使おうとしたからだ」と判断しよう。



Array は配列、Index は添え字、OutOfBounds は範囲外という意味で覚えればラクですね。

英語が表示されたからといって慌てる必要はない。どれも簡単な単語だ。逃げずに読むことが実は上達への近道だよ。



4.4

配列のデータをまとめて扱う

4.4.1 for文と配列



よし！例外が出なくなったぞ。これで完璧だ！！

最初に比べてよくなったね。でも、まだ改善できる箇所があるよ。今いまだと、将来科目が増えた場合に合計点を算出する箇所に修正が必要だろう？そのめんどうも解決できるんだよ。



実は、配列の添え字の指定に変数を用いることができます。たとえば、変数 a に 3 が入っているとき、「score[a]」という記述をすれば、前から 4 番目の要素にアクセスできるのです。

このことを利用して、次のリスト 4-9 のようなことができます。

リスト 4-9 配列と for 文

```

1 public class Main {
2     public static void main(String[] args) {
3         int[] score = { 20, 30, 40, 50, 80 };
4         for (int i = 0; i < score.length; i++) {
5             System.out.println(score[i]);
6         }
7     }
8 }
```

Main.java

score[0],score[1]…

変数 i に注目してください。for ループにより、i は、 $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ と

変化していくので、score の添え字に変数 i を用いることで score[0] から score[4] を表示させることができます。このように最初の要素から最後の要素まで、代入されている値を順番に使用することを「配列を回す」ともいいます。

そして、3 行目で作成している要素の数が 6 つや 7 つに変わっても、この「全内容の表示部分」はいっさい修正が不要な点に注目してください。**for 文の繰り返し条件に配列変数名 .length を用いる**ところがポイントです。

このように for 文を用いて配列の各要素の値を順番に取り出し、それに対して何らかの処理を行うコードを書く機会は多いので、必ず身に付けておきましょう。



配列を for ループで回す

```
for ( int i = 0 ; i < 配列変数名 .length ; i++ ) {  
    :  
}
```

4.4.2 拡張 for 文

拡張 for 文は、配列の要素を 1 つずつ取り出すループを簡単に書くために導入された新しい for 文の文法です。



拡張 for 文で配列を回す

```
for ( 要素の型 任意の変数名 : 配列変数名 ) {  
    :  
}
```

拡張 for 文では、ループが 1 周するたびに次の要素の内容が「任意の変数名」で指定した変数に格納されるため、ブロックの中ではその変数を利用して要素を取り出します。従来の for 文と比較しながら次のコードを見てください。

リスト 4-10 従来の for 文での例

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int[] score = { 20, 30, 40, 50, 80 };  
4         for (int i = 0; i < score.length; i++) {  
5             System.out.println(score[i]);  
6         }  
7     }  
8 }
```

Main.java

リスト 4-11 拡張 for 文の例

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int[] score = { 20, 30, 40, 50, 80 };  
4         for (int value : score) {  
5             System.out.println(value);  
6         }  
7     }  
8 }
```

Main.java

拡張 for 文を使うと、コードにループ変数や添え字などが登場しなくなり、すっきりします。

4.5

配列の舞台裏

4.5.1 配列を理解する



配列も回せるようになったし、これで配列は完璧ですね！

4
章

確かに基本的な内容は十分マスターしたね。でも、まだ知つておくべきことがあるよ。次のコードを見てごらん。最後に何が
出力されると思う？



リスト 4-12 実行結果は？

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int[] a = { 1, 2, 3 }; a[0] は 1  
4         int[] b;  
5         b = a;  
6         b[0] = 100; b[0] に 100 を代入  
7         System.out.println(a[0]);  
8     }  
9 }
```

Main.java

a[0] を表示すると…

「1」が出力されると思いましたか？しかし、実際に出力されるのは「100」なのです。

なぜ「100」が表示されるかを理解するために、この節では「配列を利用しているとき、コンピュータの中では何が起きているか」という舞台裏を解説します。この舞台裏を十分理解できているかどうかで、今後の章における理解や応用力に格段の差が付きます。やや高度な内容ですが、ぜひ理解してください。

4.5.2 メモリと変数

これまで p.142 の図 4-2 のように、配列は「～[] 型の配列変数に～型の要素を入れて作成する」というイメージを示してきました。実は、これは私たち人間に理解しやすくするためにイメージ図であって、実際にコンピュータの中でこのような構造になっているわけではありません。

コンピュータは使用するデータをメモリに記録します。メモリの中は碁盤の目のように区画整理されており、各区画には住所(アドレス)が振られています。そして変数を宣言すると、空いている区画(どこが選ばれるかわからない)を変数のために確保します(変数の型によって何区画を使用するかは異なります)。変数に値を代入するとは、確保しておいた区画に値を記録することなのです。

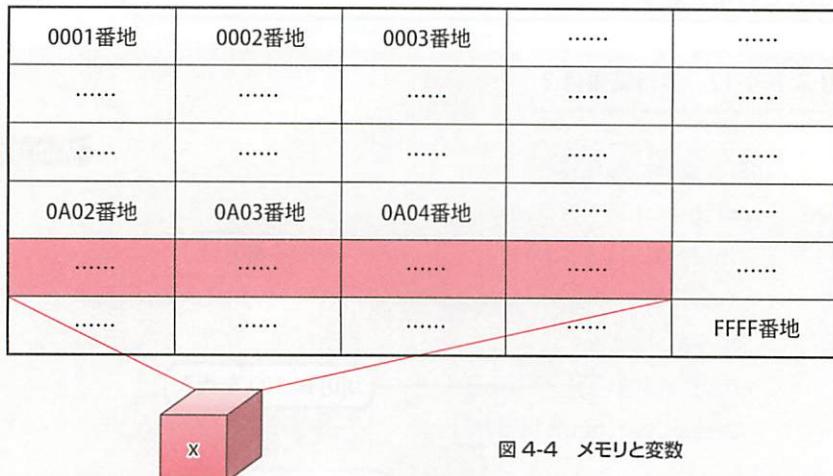


図 4-4 メモリと変数

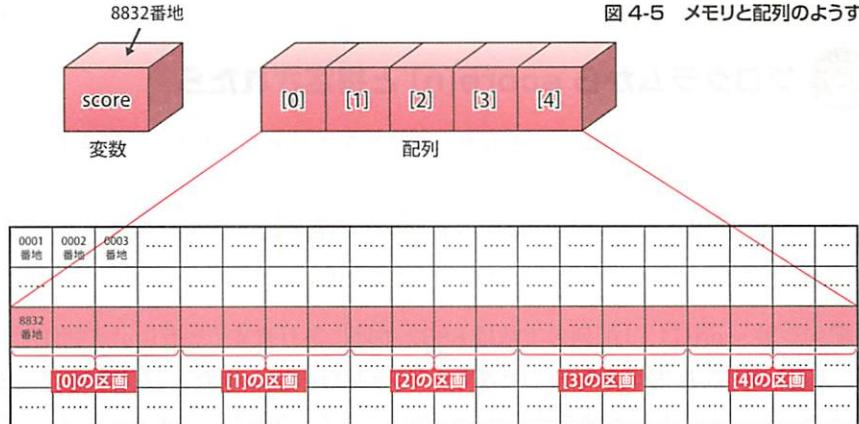
int型 (変数1つで4バイト消費する。p.48参照)

4.5.3 メモリと配列

では、配列を作成(「`int [] score = new int[5];`」)したとき、メモリの中ではどのようなことが起きているのでしょうか。

配列変数の宣言により `int[]` 型変数が、`new` 演算子により配列の実体(要素の集まり)がメモリ上の区画に作成されます(図 4-5)。そして、配列変数には 5 つ

図 4-5 メモリと配列のようす



の要素まるごとではなく、「最初の要素のアドレス」が代入されます。



int[] score = new int[5]; を実行したときのメモリ上のようす

- ① int 型の要素を 5 つ持つ配列がメモリ上に作成される。
- ② int[] 型の配列変数 score がメモリ上に作成される。
- ③ score に配列の先頭要素のアドレスが代入される。



int[] 型だけでなく、double[] 型や String[] 型の配列変数も配列の先頭要素のアドレスを代入するんですね。

そうだよ。ただし、int[] 型の配列変数に、要素が double 型である配列の先頭要素のアドレスは代入できない。int[] 型の配列変数は、要素が int 型である配列の先頭要素のアドレスしか代入できないんだ。



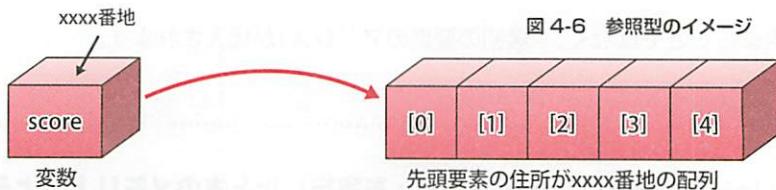
Java は次のようなしくみで配列の要素を利用しています。



プログラムから score[n] と指定されたら

- ① score の中に入っている番地 (=8832) を取り出し、配列(先頭要素)を見つける。
- ②見つけた配列の先頭要素から n 個後ろの要素の区画を読み書きする。

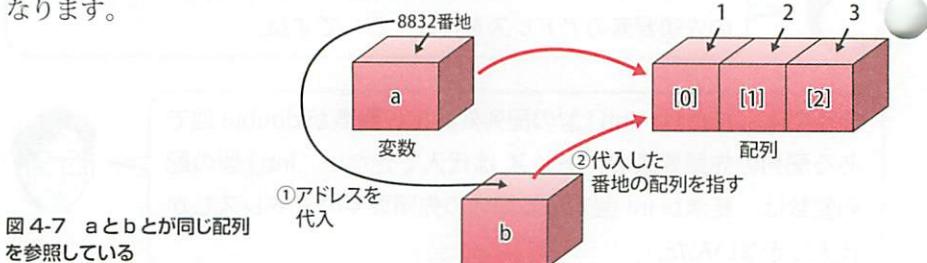
配列変数 score は、「配列の実体は 8832 番地にあります」と指示する動作をしていることになります。このことを「参照」と呼びます。また、具体的なデータではなく、メモリ上の番地を代入する変数のことを「参照型」(reference type)変数といい、int や boolean のような「基本型」変数と区別します。



4.5.4 配列を複数の変数で参照

ここまで学んだ内容を十分に理解すれば、先ほどのリスト 4-12 ではどのような状況になっているかが想像できるでしょう。

変数 a に番地「8832」が入っているとした場合、5 行目でコピーされているのは、この 8832 になります。その結果、**変数 b は変数 a と同じ配列を参照することになります。**



よって、a[0] と b[0] は、まったく同一のものを指していることになります。この状態で「b[0] = 100;」を実行することは「a[0]=100;」と同じです。当然、7 行目では 100 が output されます。

4.6 配列の後片付け

4.6.1 ガベージコレクション

次のコードを見てください。

4
章

リスト 4-13 ガベージコレクション

```
1 public class Main {  
2     public static void main(String[] args) {  
3         boolean b = true;  
4         if (b == true) {  
5             int[] i = { 1, 2, 3 };    }  
6     }  
7 }  
8 }
```

if ブロック内で配列を作成

5行目で、配列変数 `i` が宣言され、同時に 3 つの要素を持つ配列が生成されています。しかし、変数の寿命は自分が宣言されたブロックが終了するまで(3.2.1項)だったことを思い出してください。これは配列変数でも同じです。

つまり、6行目の時点で配列変数 `i` はメモリから消滅します。一方、`new` で確保された要素たちは普通の変数ではありませんので、ブロックが終了しても寿命は迎えません。その結果、配列はどの配列変数からも参照されない状態でメモリに残ってしまいます。

残った配列は、Java のプログラムからどのような方法を使っても読み書きすることはできず、事実上メモリ内のゴミ (garbage) となります。ゴミとなってしまった配列を放置し続けると、こういったゴミが溜まり続け、メモリを圧迫してしまう可能性があります。

本来ならば、「使用できなくなった配列は、もう使いませんから、破棄して(区)

画から取り除いて) メモリ領域をお返します」というメモリの後片付けをプログラマが行わなければなりません。

しかし、Java にはガベージコレクション(GC: garbage collection)というしくみが常に動いており、実行中のプログラムが生み出したメモリ上のゴミ(=どの変数からも参照されなくなったメモリ領域)を自動的に探し出して片付けてくれます。



勝手にゴミを探して片付けてくれるなんて自動掃除機みたい。楽でいいわね♪

4.6.2 null

先の項では変数の寿命によって配列変数が配列を参照しなくなる例でしたが、**null**を使用することで、意図的に参照されないようにすることができます。次のコードを見てください。

リスト 4-14 null の利用

```

1 public class Main {
2     public static void main(String[] args) {
3         int[] a = { 1, 2, 3 };
4         a = null; // 配列変数 a に null を代入
5         a[0] = 10;
6     }
7 }
```

Main.java

上記の 4 行目では配列変数に null という特別な値を代入しています。null とは「何もない」という意味で、配列変数 a のような参照型の変数に代入することができます。null が代入されると、参照型の変数はどこも参照していない状態になります。このように、ある番地を参照していた配列変数に null を代入し、参照させなくすることを「参照を切る」ともいいます。



null とは？

- ① int[] 型などの参照型変数に代入すると、その変数は何も参照しなくなる。
- ② int 型などの基本型変数には代入することができない。

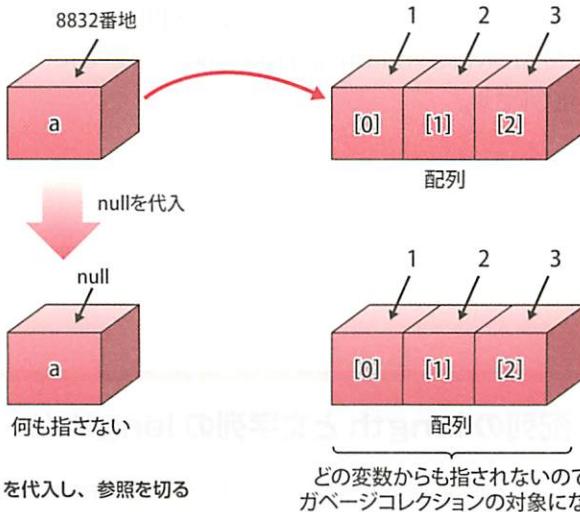


図 4-8 null を代入し、参照を切る

4.6.3 NullPointerException



先輩、さっきのプログラム、4行目で null を代入した後、5行目で「`a[0] = 10;`」していますけど、`a` はどこも参照していませんよね？



よく気づいたね。でも、コンパイルは成功するんだ。実行するとどうなるかな？



あっ、またエラーになりましたよ！ NullPointerException…例外ですね！

リスト 4-14 を実行すると、次のようなエラーが画面に表示されます。

実行結果（エラー）

```
Exception in thread "main" java.lang.NullPointerException  
at Main.main(Main.java:5)
```

この例外は、null が格納されている配列変数を利用しようとすると発生します。先ほど学習した `ArrayIndexOutOfBoundsException` とともに、配列関連で見かけることの多い例外です。



配列の `length` と文字列の `length()`

この章では、配列の要素数を取得するために「配列変数名 `.length`」という記述が可能なことを学びましたが、これと似た記述方法として、「文字列変数名 `.length()`」があります。

`String` 型の変数の後に「`.length()`」を付けることで、その文字列型変数に格納されている文字列の長さ（文字数）を得ることができます。その際には、全角／半角を問わず 1 文字としてカウントされますので、たとえば次のコードを実行すると、画面には 7 が表示されます。

```
String s = "Javaで開発";  
System.out.println(s.length());
```

配列の「`length`」と文字列型の「`length()`」は、どちらも長さを取得するためのものですが、「文字列のときは () を付ける、配列のときは () を付けない」と覚えておきましょう。

4.7

多次元の配列

4.7.1 多次元配列とは？

今まで学習してきた配列は1次元配列といいます。1次元配列に縦の並びを加えると**2次元配列**になります。

2次元配列は図4-9のように要素が縦横に並んだ表のようなものです。データを表のような形で扱いたいときに使用すると便利です。

ちなみに、2次元以上の配列を多次元配列と呼びます。ビジネスアプリケーションの開発では多次元配列を使う機会は少ないですが、科学技術計算などでは多く利用されます。

[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
[2][0]	[2][1]	[2][2]	[2][3]	[2][4]

図4-9 2次元配列のイメージ

2次元配列の宣言

要素の型 [][] 配列変数名 = new 要素の型 [行数][列数];

2次元配列の要素の利用

配列変数名 [行の添え字][列の添え字]

たとえば、兄弟2人の3科目のテスト結果を格納する2次元配列は、次のように書くことができます。

リスト 4-15 2 次元配列

```

1  public class Main {
2      public static void main(String[] args) {
3          int[][] scores = new int[2][3];
4          scores[0][0] = 40;
5          scores[0][1] = 50;
6          scores[0][2] = 60;
7          scores[1][0] = 80;
8          scores[1][1] = 60;
9          scores[1][2] = 70;
10         System.out.println(scores[1][1]);
11     }
12 }

```

Main.java

2 行 3 列の配列

このリストの 2 次元配列を図にすると下のようなイメージです。
 2 次元配列では 1 次元配列と異なり [] を 2 つ使用します。最初の [] で行、次の [] で列を指定します。

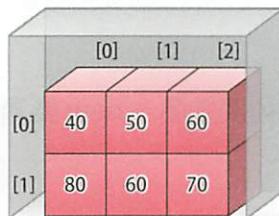


図 4-10 2 行 × 3 列の点数表

しかし、図 4-10 はあくまでもイメージです。Java における 2 次元配列は、正確には「表」ではなく、「配列の配列」になります。まず、先ほどの「表」のイメージを「配列の配列」というイメージに変化させてみましょう。2 行 × 3 列の表の場合、要素数 2 の行配列（親配列）の中に、それぞれ要素数 3 の列配列（子配列）が入ります。実際のメモリ上では、図 4-11 のようになっています。

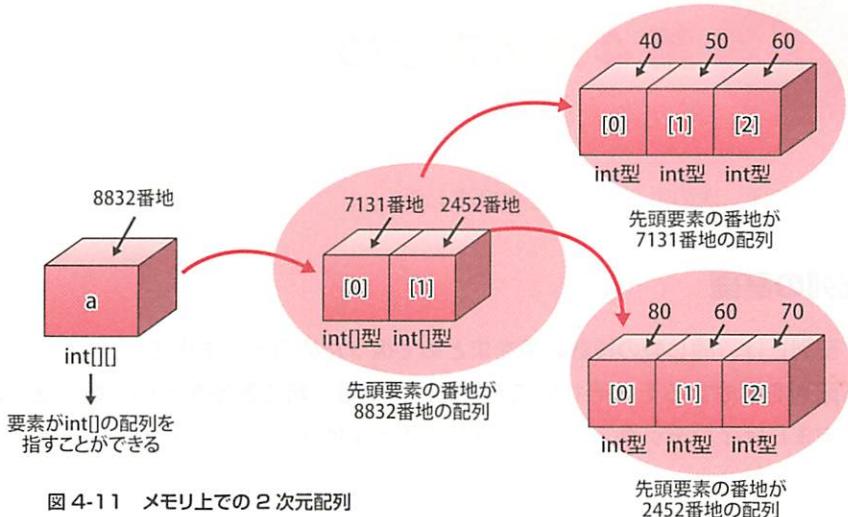


図 4-11 メモリ上の 2 次元配列

この図 4-11 の状態は次のプログラムで確認できます。

リスト 4-16 親配列と子配列の要素数を表示

```

1 public class Main {
2     public static void main(String[] args) { このような初期化が可能
3         int[][] scores = { { 10, 20, 30 }, { 30, 40, 50 } };
4         System.out.println(scores.length); 2 が出力される
5         System.out.println(scores[0].length); 3 が出力される
6     }
7 }
```

4.8

第4章のまとめ

この章では、次のようなことを学びました。

配列の基礎

- ・配列とは、同じ型の複数の値をまとめて扱うためのデータ構造。
- ・配列を構成するそれぞれの箱を要素、何番目の箱であるかという数字を添え字またはインデックスという。配列の添え字は 0 から始まる。

配列の準備

- ・配列を利用するためには、「配列変数の宣言」「要素の作成」という 2 つのステップで配列を準備しなければならない。
- ・配列変数の型には「要素の型 [] 」を指定する。
- ・要素を作成するには、「new 要素の型 [要素数] 」とし、配列変数に代入する。

配列の利用

- ・「配列変数名 [添え字] 」でそれぞれの要素を読み書きできる。
- ・for 文や拡張 for 文を用いて配列要素に 1 つずつ順番にアクセスする。

配列と参照

- ・配列変数は、配列の実体 (new で確保された各要素のメモリ領域) を参照している。
- ・特別な値 null が代入された配列変数は、どの実体も参照しない。
- ・何らかの理由で参照されなくなったメモリ領域は、ガベージコレクションによって自動的に解放される。

4.9**練習問題****練習 4-1**

次の条件に合った各配列を準備するプログラムを作成してください。なお、以下の4つの条件のコードを1つのプログラムの中に記述して構いません。また、値の初期化は不要です。

- ① int 型の値を4個まとめて格納できる配列 points
- ② double 型の値を5個まとめて格納できる配列 weights
- ③ boolean 型の値を3つまとめて格納できる配列 answers
- ④ String 型の値を3つまとめて格納できる配列 names

練習 4-2

次に示す3つの条件に沿ったプログラムを作成してください。

- ① 3つの口座残高「121902」「8302」「55100」が格納されているint型配列 moneyList を宣言します。
- ② その配列の要素を1つずつfor文で取り出して画面に表示させます。
- ③ 同じ配列の要素を拡張for文で1つずつ取り出して画面に表示します。

練習 4-3

次のコードを実行すると、5行目と6行目で例外が発生します。それぞれの行で発生する例外の名前を答えてください。

```

1 public class Main {
2     public static void main(String[] args) {
3         int[] counts = null;
4         float[] heights = { 171.3F, 175.0F };
5         System.out.println(counts[1]);

```

Main.java

```
6     System.out.println(heights[2]);  
7 }  
8 }
```

練習 4-4

次に示す 4 つの条件に沿って「数あてクイズ」のプログラムを作成してください。

- ① int 型で要素数 3 の配列 numbers を準備します。このとき初期値はそれぞれ「3」「4」「9」とします。
- ② 画面に「1 行の数字を入力してください」と表示します
- ③ 次のコードを用いてキーボードから数字の入力を受け付け、変数 input に代入します。

```
int input = new java.util.Scanner(System.in).nextInt();
```

- ④ 配列をループで回しながら、いずれかの要素と等しいかを調べます。もし等しければ「アタリ！」と表示します。

4.10 練習問題の解答

練習 4-1 の解答

以下は解答例です（おおむね合っていれば正解として構いません）。

```
1 public class Main {
2     public static void main(String[] args) {
3         int[] points = new int[4];
4         double[] weights = new double[5];
5         boolean[] answers = new boolean[3];
6         String[] names = new String[3];
7     }
8 }
```

Main.java

4
章

練習 4-2 の解答

以下は解答例です（おおむね合っていれば正解として構いません）。

```
1 public class Main {
2     public static void main(String[] args) {
3         int[] moneyList = { 121902, 8302, 55100 };
4         for (int i = 0; i < moneyList.length; i++) {
5             System.out.println(moneyList[i]);
6         }
7         for (int m : moneyList) {
8             System.out.println(m);
9         }
10    }
11 }
```

Main.java

練習 4-3 の解答

5 行目 : NullPointerException

6 行目 : ArrayIndexOutOfBoundsException

練習 4-4 の解答

以下は解答例です（おおむね合っていれば正解として構いません）。

```
1 public class Main {  
2     public static void main(String[] args) {  
3         // (1)配列の準備  
4         int[] numbers = { 3, 4, 9 };  
5  
6         // (2)メッセージの表示  
7         System.out.println("1桁の数字を入力してください");  
8  
9         // (3)キーボードからの数字入力  
10        int input = new java.util.Scanner(System.in).nextInt();  
11  
12        // (4)配列を回しながら判定  
13        for (int n : numbers) {  
14            if (n == input) {  
15                System.out.println("アタリ！");  
16            }  
17        }  
18    }  
19 }
```

Main.java