配列とは

独立したデータであれば、それで問題ありません。しかしプログラミングでは、関連する複数のデータをまとめて管理したいケースもあります。

たとえば、5人分のテストの点数をまとめて管理したい場合を考えましょう。各々の点数を単独の変数で管理しようとすると、以下のように書くことになりますね。

int score1 = 70;

int score2 = 85;

int score3 = 80;

int score4 = 95;

int score5 = 90;

仮に人数が100人に増えたら、上記のような変数をあと95個も用意しないといけません。プログラミングが大変なのはもちろん、変数の関連性もわかりづらくなります。

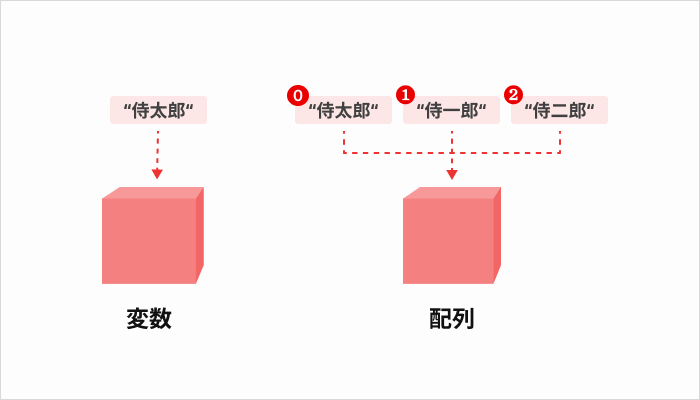
こうしたケースで役に立つのが 「配列」 です。配列を使えば、以下のように関連する複数のデータを1つの変数でまとめて管理できます。

int[] scoreArray = { 70, 85, 80, 95, 90 };

配列は効率的なプログラムを作るうえで重要な文法の1つです。

配列とは簡単にいえば、複数のデータをまとめたものです。配列を使うことで、データ型が同じ複数のデータを1つの変数のように扱えます。

たとえば、「侍太郎」「侍一郎」「侍二郎」の3つを管理するとしましょう。これまでの変数だと1データずつしか管理できませんが、配列なら1つの変数でまとめて管理できます



配列で複数のデータを扱う場合、上図のように0、1、2……と各データに「0」から始まる番号を割り振ります。この番号が 「インデックス」 です。

また、配列に入れる各データ（侍太郎、侍一郎、侍二郎）を 「要素」 と呼びます。詳しくは後述しますが、Javaの配列ではインデックスを用いて各要素を参照します。

なお、整数の「123」と文字列の「ABC」など、データ型の異なるデータは1つの配列にはできません。配列は、データ型が同じ複数データをまとめて扱える変数と理解しましょう。

配列の使い方を理解しよう

まずは、配列の基本的な使い方を理解しましょう。配列も変数の一種のため、配列を使うときにも単独の変数と同様の手続きが必要となります。

変数を使うときの流れは、宣言→代入→参照が基本

宣言（箱を用意する）

代入（箱にデータを入れる）

参照（箱のデータを使う）

配列でも、この順序は基本的に変わりません。

配列の宣言

まずは、配列の「箱」を用意するために宣言が必要となります。Javaの配列には、大括弧 [ ] を使うのが特徴です。データ型または配列名に[ ]をつけると、配列の宣言と扱われます。

データ型[] 配列名; // 書き方1（推奨）

データ型 配列名[]; // 書き方2

データ型に[ ]をつける前者の書き方を推奨します。 配列名が長くなっても、行の頭を見るだけで配列だとひと目でわかるためです。

たとえば、int型の配列「scoreArray」を宣言するときは、以下のように書きます。なお配列名には、「Array（配列）」という英単語が使われることが多いです。

int[] scoreArray;

ただし、上記では「こんな配列を使いますよ」と宣言しただけで、実際の箱は用意されません。そもそも何個の箱が必要かわからないためです。

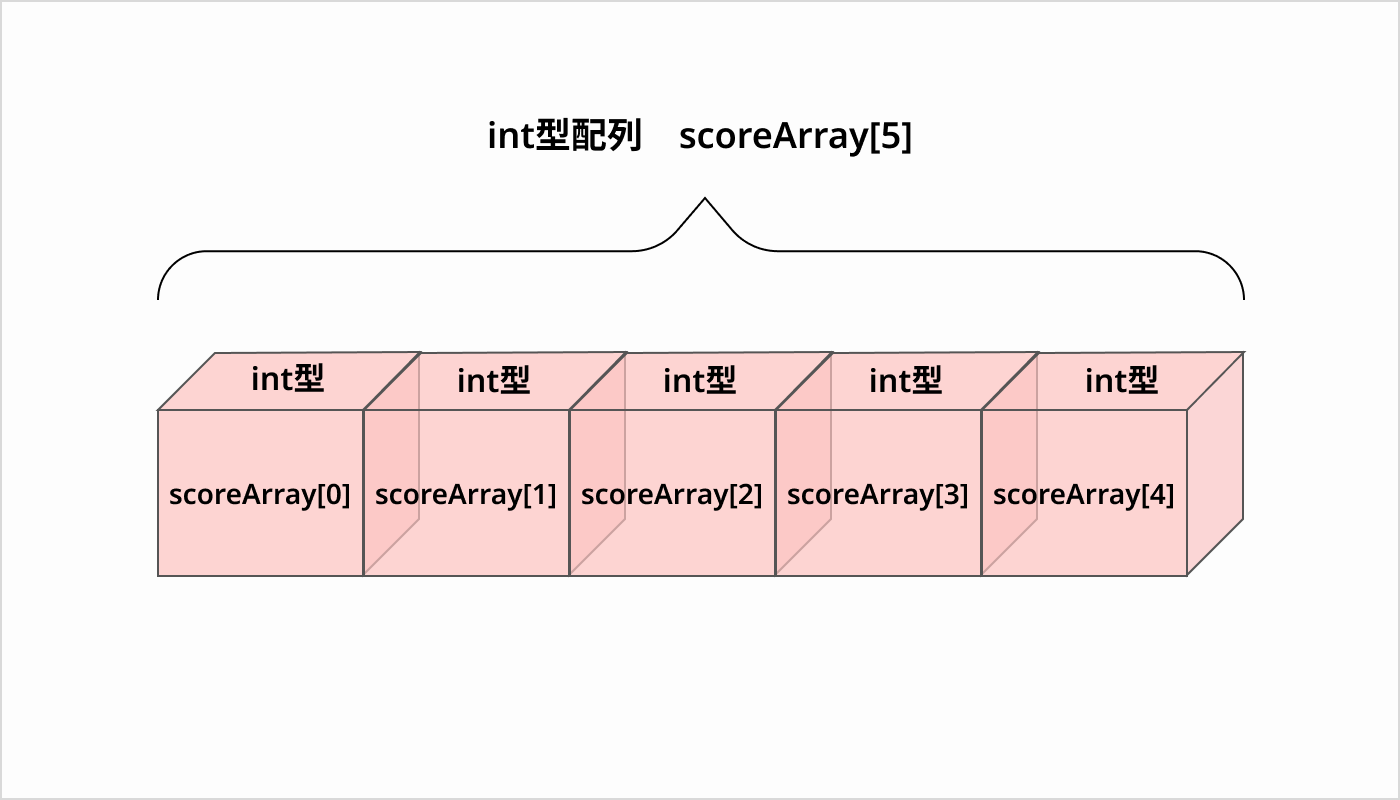
配列の箱を用意するために、new演算子を用います。new演算子は、配列やクラスの「実体」をつくり出すためのものです。配列の箱を用意するときは、以下のように書きます。

データ型[] 配列名 = new データ型[要素数];

「要素数」には、管理したいデータの数を指定してください。たとえば、5人分のテストの点数を管理するなら、以下のように宣言します。

int[] scoreArray = new int[5];

こうすることで、int型のデータを入れる箱が5個だけ作られます。配列の各データは以下のように1つにまとまっているため、データの管理が容易です。



配列の各データには、上図のように[0]～[4]のインデックスが割り振られます。インデックスは、マンションの部屋番号のようなイメージです。0から始まる点に注意しましょう。

なお、new演算子で配列の箱を作った場合、各要素の初期値は以下のようになります。

データ型 初期値

整数型 0

浮動小数点数型 0.0（実質0）

論理型 false

文字列型 null（後述）

文字列型の 「null」 は、「データが何も入っていない状態」のことです。整数型は「0」というデータが入っていますが、文字列型ではそもそもデータが空の状態をnullと扱います。

単独変数を初期化せずに宣言した場合、初期値が何も設定されないため、そのままでは参照できません。一方、new演算子で作った配列は初期値が設定され、そのまま参照できます。

配列の代入・初期化

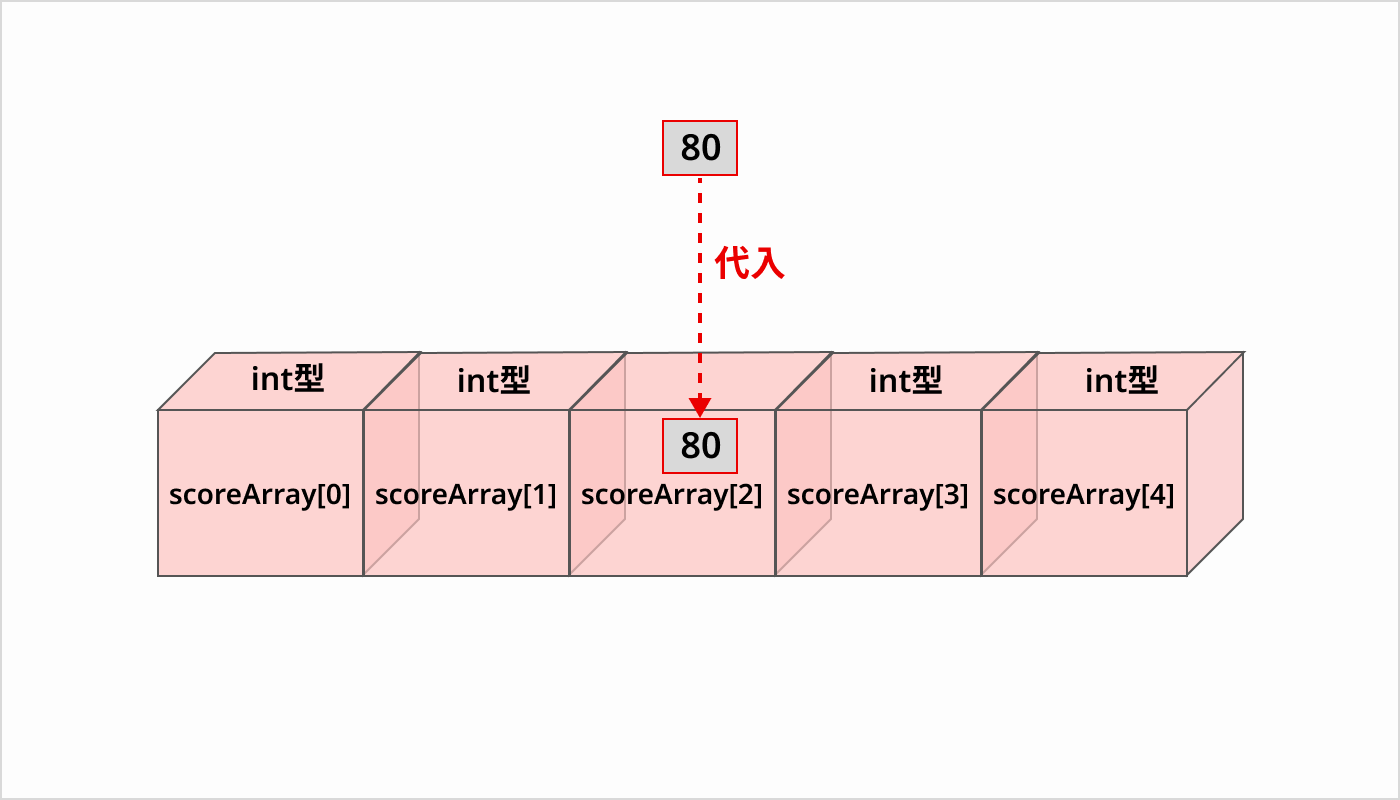
配列の箱を作ったら、値を代入します。宣言した配列の各要素にデータを代入するときの基本的な書き方は、以下のとおりです。

配列名[インデックス] = 値;

配列名にインデックスを指定することで、1つの要素に値を代入できます。たとえば、配列scoreArrayの3番目の要素に「80」を代入するときの書き方は、以下のとおりです。

scoreArray[2] = 80;

前述のとおり、インデックスは0、1、2……のように0から始まります。そのため、3番目の要素番号は「2」となる点に注意しましょう。上記の処理イメージは以下のとおりです。



また、単独変数と同様に初期化することも可能です。初期化は、宣言時に初期値も代入することでしたね。最初から配列の値が決まっているときは、以下のように初期化しましょう。

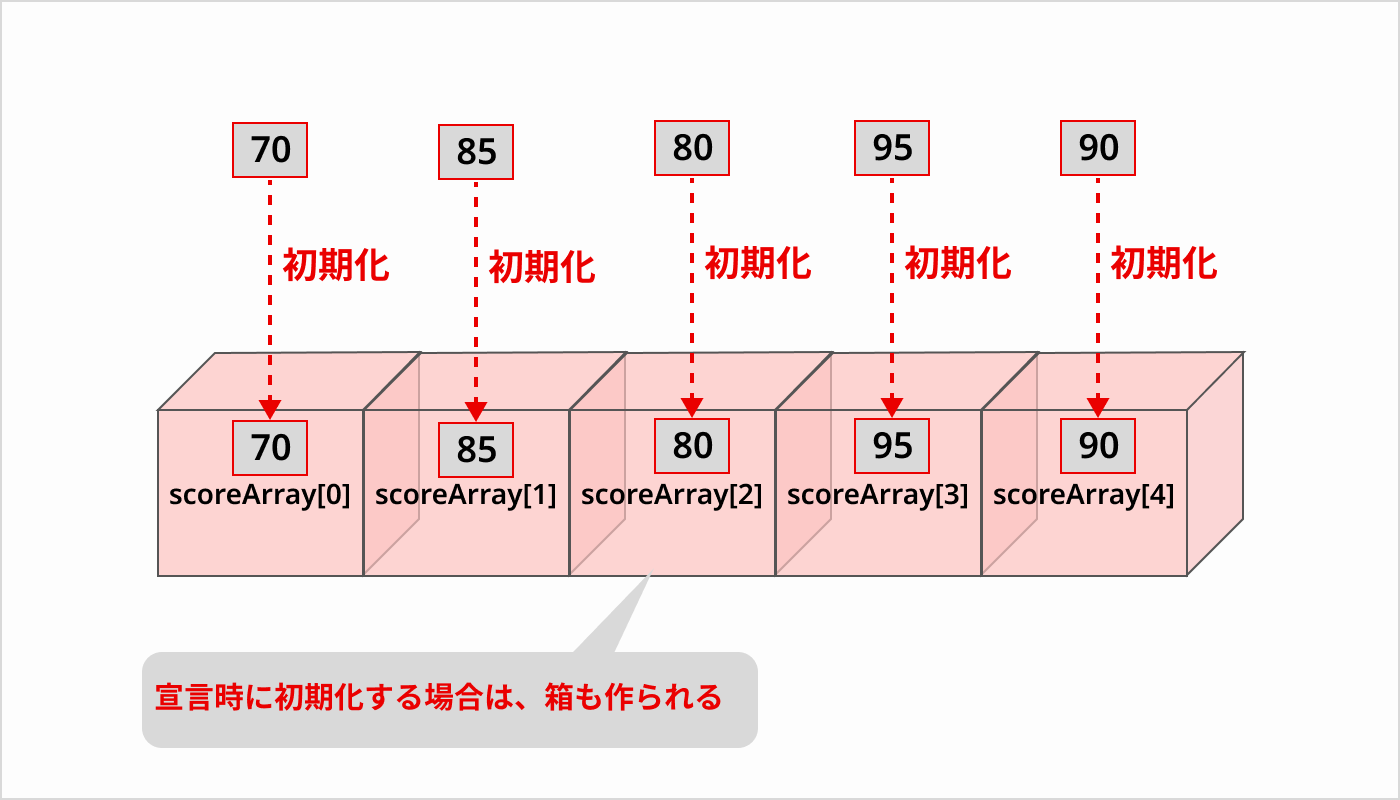
// n個の要素を持つ配列の初期化

データ型[] 配列名 = { 要素1の値, 要素2の値, 要素3の値,・・・, 要素nの値 };

配列の初期化では、中括弧{ }の中にカンマ（,）区切りで各要素の初期値を順番に書いてください。たとえば、5人分のテストの点数で初期化するときの記述例は、以下のとおり。

int[] scoreArray = { 70, 85, 80, 95, 90 };

初期化つきで配列を宣言する場合、初期値を設定した要素数分の箱も作られます。上記のケースでは5個分の箱が作られるため、前述のnew演算子を使う必要はありません。



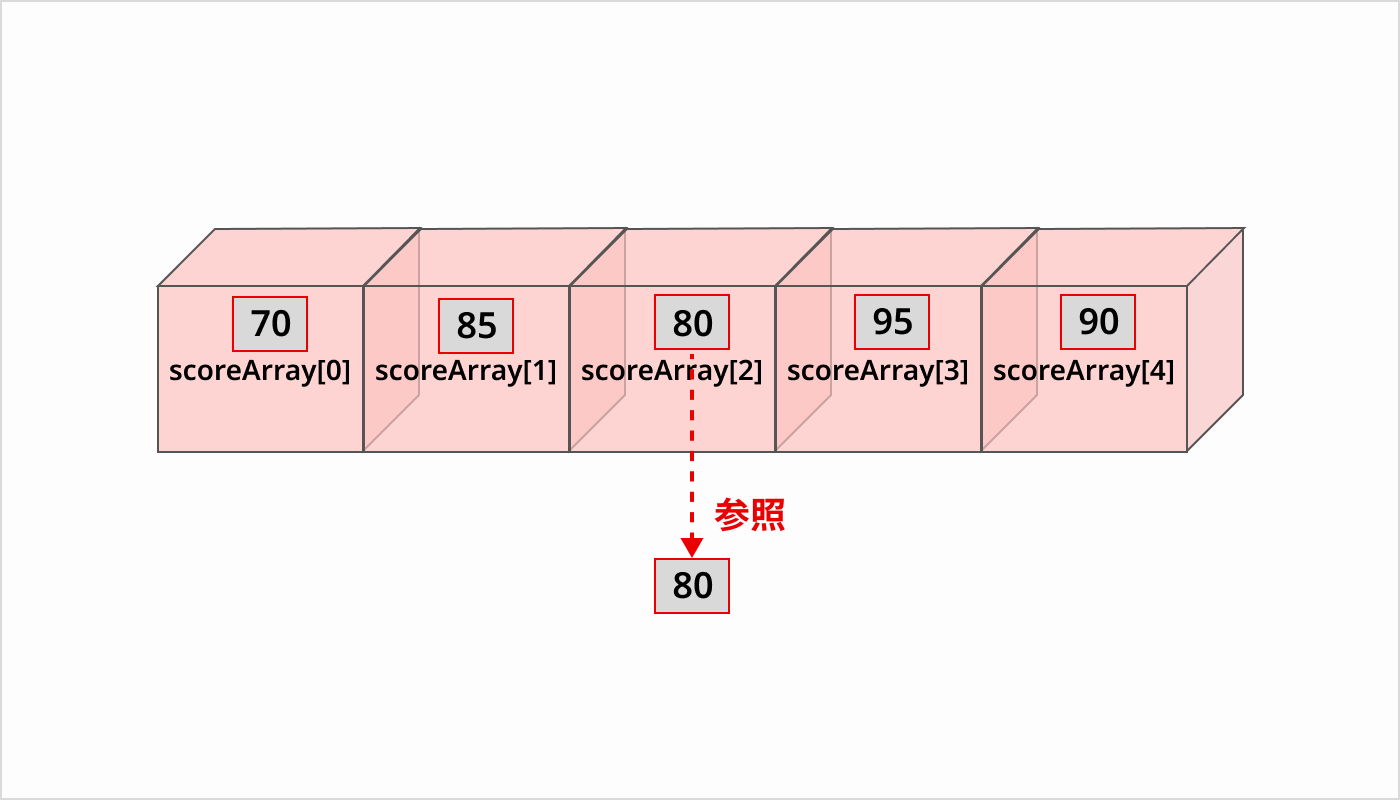
配列の参照

配列に値を代入したら、参照します。参照とは、箱の中身をチェックすることでしたね。配列名に、参照したい要素のインデックスを添えることで、配列のデータを参照できます。

配列名[インデックス]

たとえば、if文で「配列scoreArrayの3番目のデータが90未満か」と判定する場合、以下のように書きます。繰り返しになりますが、インデックスが0から始まることに注意。

if( scoreArray[2] < 90 ) {



先ほど3番目の要素に「80」を代入したため、「90未満か」の条件式はtrue（成り立つ）と判定されます。単独変数の参照と同様に、値を見るだけで中身自体はなくなりません。

配列を使ってみよう

では、実際に配列を使ってみましょう。今回は、5人分の氏名・点数をそれぞれ配列にして表示します。以下のようにソースコードを書いてください。

package text.section11;

public class Array01 {

public static void main(String[] args) {

String[] nameArray = new String[5]; // 氏名

int[] scoreArray = { 70, 85, 80, 95, 90 }; // 点数

// 氏名を代入

nameArray[0] = "侍太郎";

nameArray[1] = "侍一郎";

nameArray[2] = "侍二郎";

nameArray[3] = "侍三郎";

nameArray[4] = "侍四郎";

// 氏名を出力

System.**out**.println(nameArray[0]);

System.**out**.println(nameArray[1]);

System.**out**.println(nameArray[2]);

System.**out**.println(nameArray[3]);

System.**out**.println(nameArray[4]);

// 点数を出力

System.**out**.println(scoreArray[0]);

System.**out**.println(scoreArray[1]);

System.**out**.println(scoreArray[2]);

System.**out**.println(scoreArray[3]);

System.**out**.println(scoreArray[4]);

}

}

氏名を管理する配列nameArrayは、new演算子を用いて生成し、各要素にデータを代入しています。一方、点数を管理する配列scoreArrayは、中括弧を用いて初期化しています。

作成したプログラムを実行してみましょう。以下のように、5人分の氏名・点数が順番に表示されることを確認してください。

侍太郎

侍一郎

侍二郎

侍三郎

侍四郎

70

85

80

95

90

このように配列を使うことで、複数データを1つの変数のようにまとめて使えます。

補足：要素外のデータ参照に注意しよう

配列の利用時には、要素外のデータ参照に注意しましょう。たとえば、5個分の箱しか作っていないのに、6番目の要素を参照することはできません。

これまで「インデックスは0から」と何度も言ってきたのには、意味があります。インデックスが0から始まるという意識がないと、以下のようなミスをしやすいためです。

int[] scoreArray = { 70, 85, 80, 95, 90 }; // 点数

// 配列scoreArrayは[0]～[4]のため、[5]は参照できずエラーになる

System.out.println(scoreArray[5]);

配列scoreArrayは、初期化時に5個の要素を生成しています。前述のように、5個の要素で参照できるのは[0]～[4]のため、scoreArray[5]としても参照できる箱がないのです。

上記のプログラムをEclipseで実行すると、以下のようにエラーが出てしまいます。



プログラムを書き間違えても、大概はEclipseが実行前にエラーを出してくれます。しかし、 配列の要素外参照は実行時に初めてエラーが出るため注意 が必要です。

配列の要素外参照を防ぐうえでは、後述する拡張for文が役立ちます。

また、配列の要素外参照を始めとする不測の事態に備えられる、「例外処理」 というテクニックも便利です。

for文の繰り返し処理で配列を有効活用しよう

繰り返し処理のfor文は、配列と相性が良い構文です。配列の各要素を順番に参照したいときには、for文を使うことでスマートに実装できます。

これまでのfor文を使う場合

たとえば、先ほどの配列scoreArrayの要素[0]～[4]を順番に表示するとしましょう。これまでに学んだfor文を用いると、以下のように3行だけで済みます。

for( int i = 0; i < scoreArray.length ; i++ ) {

System.out.println(scoreArray[i]);

}

実行結果

70

85

80

95

90

「scoreArray.length」は配列scoreArrayの要素数、つまり「5」を意味します。上記のように、配列の要素数を処理に使いたいときは、以下のように書きましょう。

配列名.length

上記のfor文は、「変数ｉが0から始まり、要素数の5を超えない範囲（4）まで繰り返す」となります。ｉを配列のインデックスにすれば、[0]～[4]まで順番に参照できるのです。

では、実際に試してみましょう。ここでは、配列scoreArrayの各要素のデータをfor文で表示した後で、要素数も表示します。以下のようにソースコードを書いてください。

package text.section11;

public class Array02 {

public static void main(String[] args) {

// **TODO** 自動生成されたメソッド・スタブ

int[] scoreArray = { 70, 85, 80, 95, 90 }; // 点数

// scoreArray[0]～[4]の値を順番に表示

for( int i = 0; i < scoreArray.length ; i++ ) {

System.**out**.println( (i + 1) + "番目は" + scoreArray[i] );

}

System.**out**.println("要素数は" + scoreArray.length);

}

}

以下のように、各要素[0]～[4]のデータと、要素数が表示されればOKです。

1番目は70

2番目は85

3番目は80

4番目は95

5番目は90

要素数は5

配列の要素を順番に参照したいときは、このようにfor文を使うと便利です。また「配列名.length」で要素数を取得できることも覚えておくと、要素外参照の防止に役立ちます。

拡張for文も知っておこう

前項のようなfor文も便利ですが、さらに便利な拡張for文も知っておきましょう。拡張for文は、配列などの各要素を順番に参照するために、for文をより単純化した構文です。

拡張for文の書き方は、以下のとおり。for文の初期化式や条件式、変化式はありません。

代わりに「ループ用変数の宣言」と「配列名」を、コロン（:）区切りで書きます。

for( データ型 ループ用変数名 : 配列名 ) {

反復処理;

}

ループ用変数は、「配列の各要素のデータを受け取る変数」です。拡張for文は、コロンの右側に指定した配列の要素を先頭から参照し、順番にループ用変数に代入します。

たとえば、先ほどの配列scoreArrayの各要素を順番に表示したい場合、拡張for文を用いて以下のような書き方が可能です。

// scoreArrayの各要素を順番に表示

for( int score : scoreArray ) {

System.out.println(score);

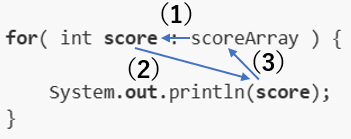
}

上記の拡張for文は、大まかに以下のような処理を繰り返します。

配列scoreArrayの要素を1つ取り出し、ループ用変数scoreに代入

データを代入した変数scoreを用いて、反復処理を行う

1に戻り、次の要素をループ用変数scoreに代入



反復処理のたびに、変数scoreには配列scoreArrayの[0]～[4]が順番に代入されます。その値を反復処理で使うことで、配列の要素を順番に参照できるのです。

では、拡張for文を試してみましょう。以下のようにソースコードを書いてください。

package text.section11;

public class Array03 {

public static void main(String[] args) {

// **TODO** 自動生成されたメソッド・スタブ

int[] scoreArray = { 70, 85, 80, 95, 90 }; // 点数

// scoreArrayの各要素を順番に表示

for( int score : scoreArray ) {

System.**out**.println(score);

}

}

}

拡張for文のメリットは、主に以下の2つです。

要素外参照を防げる

無限ループを防げる

拡張for文は、上記のように必ず存在する要素の範囲内で繰り返しが行われます。 そのため、誤ってscoreArray[5]のような要素外を参照してしまうミスを防げるのです。

また、通常のfor文だと条件式がfalseにならず、無限ループしてしまうリスクがあります。一方、拡張for文は必ず要素数だけ繰り返されるため、無限ループの心配もありません。

このように、拡張for文は便利なだけでなく、安全性も高いといえます。配列の要素を順番に参照したい場合、拡張for文を有効活用しましょう。