7-5.配列・多次元配列

配列宣言のルール

配列の宣言は、以下のような形でした。

```
//int型の配列変数arrayを宣言し、3つのint型の値を扱う配列インスタンスを代入
int[] array = new int[3];
```

newがついているということは、配列もインスタンスを作成しているという認識を持っておきましょう。
インスタンス化をする際の数字は、その配列の要素数(入る値の数)を指しました。
変数にその数字が直接代入されるわけではないので注意しましょう。

配列型変数の宣言ですが、大カッコ[]を付ける位置は、実はデータ型の後でも、変数名の後でも可能です。
これは、ほかのプログラミング言語から移行してくる技術者のための仕様です。
コンパイルエラーなどは起きないので、注意しておきましょう。

```
int[] array; //OK
int array[]; //OK
```

配列のインスタンス時、要素数を指定することも出来ました。
では、以下はどうなるでしょうか？

```
int[] array = new int[0]; //要素数0の配列インスタンスを作成
int[] array = {}; //上記と全く同じ意味
```

配列は、複数の値を扱うためのものです。
要素数ゼロを指定するということは、配列として意味を成していませんね。

しかし、この記述は文法的には何も問題ありません。
役目としてはほぼ何も果たしませんが、コンパイルエラーなどは起きないので覚えておきましょう。
また、最初から値を代入する時は波カッコ{}を使って値を代入しますが、この波カッコの中に何も書かない記述は、new int[0]と全く同じ意味を持ちます。
また、この波カッコは **初期化子** と呼びます。

次に、コンパイルエラーが起きる宣言のパターンを覚えましょう。

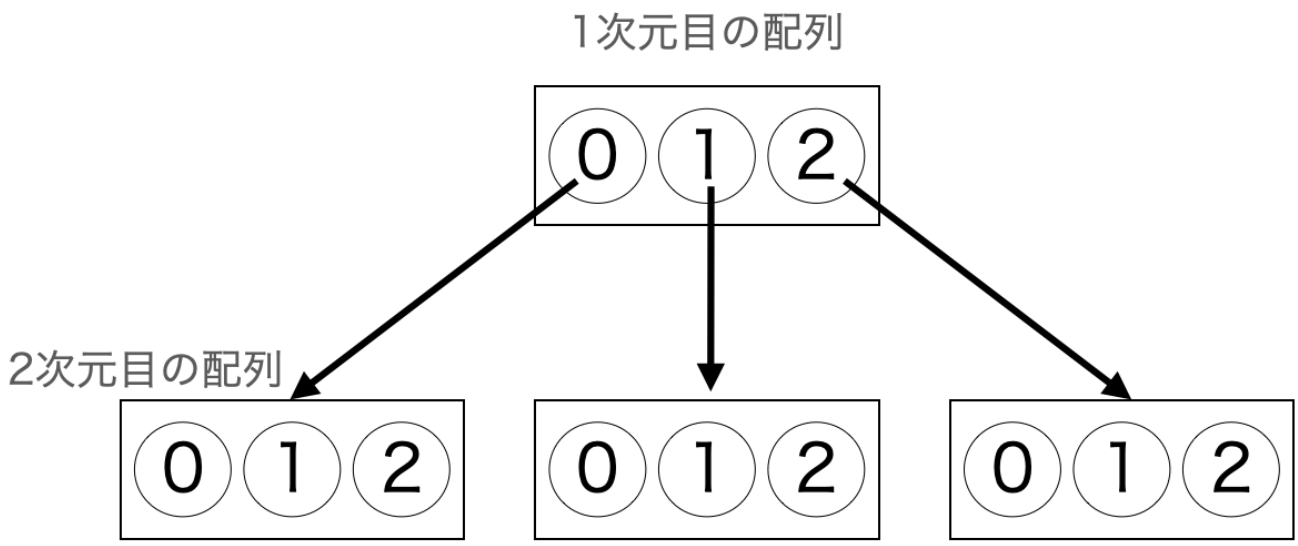
```
int[] array = new int[]; //要素数の指定無し　コンパイルエラー
int[] array = new int[2.3]; //要素数が小数など　コンパイルエラー
```

newで配列インスタンスを作成した場合は、必ず要素数を指定しなければなりません。
また、要素数は整数値(正確にはint型の範囲)でなければなりません。
これらの記述の場合は、コンパイルエラーが起きます。

多次元配列の宣言

配列の中身は、ゼロから始まるインデックス番号とセットで、1つずつ値が入っているイメージでした。
この配列の値として、また更に配列を設定することができます。
このような配列を、 **多次元配列** と呼びます。

?



上記の画像は2次元配列ですが、2次元配列に更に配列を入れた3次元配列など、次元は増やすことができます。

多次元配列変数の宣言は、以下のように行います。

```
int[][] arrayA; //2次元
int arrayB[][]; //2次元
int[] arrayC[]; //2次元
int[][] arrayD[]; //3次元
```

上記のように、配列であることを示す大カッコ`[]`を増やし、多次元の配列を表します。
また、先程記載したように、データ型の後ろでも、配列名の後ろでも宣言することができます。
さらに、多次元を示すカッコを、データ型の後ろと配列名の後ろに分けて記述することも出来ます。
多次元配列変数は、かなり柔軟に宣言出来ますね。

しかし、`new`を使用した配列インスタンス宣言の記述にはいくつか注意があります。

```
int[][] arrayA = new int[3][5]; //OK
int[][] arrayB = new int[][]; //コンパイルエラー
int[][] arrayC = new int[][5]; //コンパイルエラー

int[][] arrayD = new int[3][]; //OK
arrayD[0] = new int[5]; //後から2次元目に値を代入
arrayD[1] = new int[5];
arrayD[2] = new int[5];

int[][][] arrayE = new int[3][5][]; //OK
int[][][] arrayF = new int[3][][5]; //コンパイルエラー

int[][][] arrayG = new int[3][5]; //次元数が合っていない　コンパイルエラー
```

`arrayA`のように、それぞれの次元の要素数をはじめから宣言するのが基本の使い方です。

`arrayB`のように、どちらも要素数を指定していなければコンパイルエラーとなります。これは先程の1次元配列と同様です。

`arrayC`と`arrayD`は少し注意です。
1次元目の要素数が指定されていれば、その次の要素数は指定していなくてもOKです。
ひとまず指定せず、後で値を格納できます。

また、もっと次元が深まった場合、
`arrayE`のように、直前の要素数が指定されていれば、次の次元の要素数は指定無しにできます。
`arrayF`のように、直前の要素数が無い場合、次の次元の要素数は指定できません。

先程と合わせると、
1次元目の要素数は必須、かつ、直前の要素数が指定されていれば、次の次元の要素数は無しでも可
となります。

また、`arrayG`のように、宣言した変数とインスタンス化した配列の次元数が合っていなければコンパイルエラーとなるので注意しましょう。

配列内容の初期化

配列変数を宣言すると同時に、値を格納する時は以下のように行いました。

```
int[] array = {2, 3};
```

これは、次のようなコードでも表せます。

```
int[] array = new int[]{2, 3};
```

どちらも同じ意味を持ちますが、後者の方法は少し注意があります。
先程説明したように、newで配列のインスタンスのみ生成する時は大カッコ[]の中に要素数を指定する必要がありました。
しかし、newと初期化子{}を両方使った場合、大カッコ[]の中に要素数を指定してはいけません。

```
int[] array = new int[2]{2, 3}; //コンパイルエラー
```

初期化子に入れた要素数によって自動で計算され、さらに指定を書くと命令が被ってしまうイメージで覚えておきましょう。

多次元配列に関してもほとんど同様のルールとなります。
また多次元配列の場合、初期化子を書いた場合、要素数だけでなく次元数も自動で算出してくれる仕様があるので、明示していなくてもコンパイルエラーにはなりません。

```
int[][] array = { {2, 3}, {4, 5} }; //OK
int[][] array =new int[][]{ {2, 3}, {4, 5} }; //OK
int[][] array =new int[2][]{ {2, 3}, {4, 5} }; //コンパイルエラー

int[][] array = new int[][] {}; //OK
int[][] array = {}; //OK
```

このように、初期化子{}は便利ですが、 **変数宣言と同時にしか使えない** というルールがありますので覚えておきましょう。
初期化子は、変数宣言時の次元を元に次元を決めているため、2行に渡って記述するとエラーが出ます。
もし2行に渡って記述したい場合は、newによる配列インスタンスと大カッコで次元数を明示する必要があります。

```
int[][] arrayA;
arrayA = {}; //コンパイルエラー

int[][] arrayB;
arrayB = new int[][]{}; //OK
```

配列の仕様は非常に覚えづらくややこしいですが、試験問題ではその細かい知識を試されます！
できれば実際にコードを打ってみて、体感しながら覚えましょう。

課題

満点を取れるまで小テストを受験して下さい。

制限時間: 30 分

評定方法: 最高評点

合格点: 100 / 100

受験件数: 121