

1.2 Java 基礎

- 変数とデータタイプ
- 文字列と配列
- 演算子

目次

- 1 変数とデータタイプ
- 2 文字列と配列
- 3 演算子

Java 変数

- **変数**^[Variable]とは、プログラム中のデータ値を入れる容器です。
- Java には多くの**データタイプ**^[Type]が準備されています。例えば：
 - String : "Hello World" などの文字列。
 - int : 123 と -123 などの整数。
 - double : 19.99 や -19.99 のような浮動小数点数（小数）。
 - boolean : true と false の 2 つの状態を持つ値。

Note

Java では、各変数は **1** つのタイプしか格納できません。

変数の宣言

- 変数を**宣言**^[Declare]（作成）するには、その変数の**データタイプ**と**名前**を指定する必要があります：

```
type name;
```

- ただし、type は変数のデータタイプ（「int」や「String」）、name は変数の名前（「x」や「name」）。
- 次のコードでは、整数を格納する変数 age を宣言します：

```
int age;
```

- 同じ名の変数は**一度**しか宣言できません。

変数への代入

- **代入**^[Assignment]とは、特定のデータ値を変数に入れることです。Java では代入記号「=」を使って実現します：

```
variable = value;
```

- 代入記号は、その右側の値を左側の変数に格納します。右辺は既成の値でもいいし、算術式や他の変数、メソッドでもいいです。
- 次のコードは、整数を格納する変数 age を宣言し、そこに値 28 を代入します：

```
1 int age;
2 age = 28;
```

変数の使い方

- 一度変数を宣言し、値を代入すれば、その値は変数名によっていつでも利用することができます。
- 次のコードは、整数を格納する変数 `age` を宣言し、そこに値「28」を入れ、値を出力します：

```
1 int age;  
2 age = 28;  
3 System.out.println(age); // => 28
```

Tips 💡

変数の宣言と同時に値を代入すること（**初期化**）ができます：

```
1 String str = "Hello world!";  
2 System.out.println(str); // => Hello world!
```

変数に値が代入されると、元々格納されていた値は上書きされます：

```
1 int num = 1;  
2 num = 2;  
3 System.out.println(num); // => 2
```

名前の構成ルール

- Java では、クラスや変数などの名前（**識別子**^[Identifier]）は、開発者が自由に決めることができますが、以下の基本ルールに従う必要があります：
 1. 名前に使用できるのは、大文字と小文字のアルファベット（a～z、A～Z）、アラビア数字（0～9）、アンダースコア（_）、ドル記号（\$）のみです。
 2. 名前は数字で始めることは**できません**。
 3. 元々 Java にある**キーワード**と同じ名前にしてはいけません。
キーワード参照サイト：

 https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html

次へ 

- Java は大文字と小文字を区別します。「myVar」と「MyVar」は異なる名前です。
- Java のコードスタイルでは、「**キャメルネーミング**^[camelCase]」を使用することが推奨されています。変数名は小文字で始まり、クラス名は大文字で始まります。変数名は、簡潔で分かりやすい英単語を使用することです。

ネーミングの例

Example



- i
- string1
- studentNameList
- SomE\$oDd_name (推奨しない)

Example



- int (キーワードと重複)
- 1stVar (数字から始める)
- var#1 (不正な文字を含む)
- two words (スペースを含む)

データタイプ

- Java におけるデータタイプは 2 つに分類されます：
 - **基本データ型**は、9 種類あります。
数値のような基本的で単純なデータを格納します。
 - **参照型**には、文字列や配列、自分で定義したクラス等のデータタイプです。
 - 文字列等の複雑なデータを格納します。

データ型：データの種類

基本データ型

整数
小数
文字
真偽値

参照型

文字列
配列
クラス

基本データ型

- **基本データ型**^[Primitive Data Types]は、簡単なデータを格納するために使用されます。
- 基本データ型はすべて小文字で始まり、Java のキーワードでもあります。
- 大まかに分類すると、以下の 3 種類があります：
 - 整数型：byte、short、char、int、long；
 - 小数型：float と double；
 - ブール型：boolean。
- 他には、特殊なタイプである void もありますが、今回は割愛します。

整数型

- byte、short、char、int、long タイプは、**整数**を格納するために使用されます。
- タイプごとに数値の範囲や占有するメモリ量が異なります。

タイプ	範囲	メモリ量
byte	-128 から 127	1 Byte
short	-32,768 から 32,767	2 Byte
char	0 から 65,535	2 Byte
int	-2,147,483,648 から 2,147,483,647	4 Byte
long	-9,223,372,036,854,775,808 から 9,223,372,036,854,775,807	8 Byte

小数型

- float と double タイプは、**浮動小数点数**（小数）の格納に使用されます。
- これらの精度、範囲、占有するメモリ量などは異なります：

タイプ	精度	範囲	メモリ量
float	10 進数では約 7 桁	$-10^{-37} \sim +10^{38}$	4 Byte
double	10 進数では約 15 桁	$-10^{-307} \sim +10^{308}$	8 Byte


Note



float 型の値は「f」で終わる必要があります：

```
float a = 1.5f;
```

ブール型

- プログラミング中では、時々、2 種類の値を持つデータを使う必要があります：
 - はい / いいえ
 - オン / オフ
 - 真 / 偽
- このため、Java には true と false を格納できる boolean 型（**ブール型**）があり、true と false は**ブール値**と呼ばれます。
- 後ほど紹介する制御文（ § 1.3.1）では、ある**条件**を満たしているかどうかを判断するために、ブール値を計算する必要があります。

キャスト

- ある基本データ型の値を別の基本データ型に代入したい場合は、**型キャスト**^[Cast]する必要があります。
- Java では、2 種類のキャストがあります：
 - **拡大変換**：小さなタイプを大きなタイプに変換。
情報が失われないので、**自動的**に行うことができます。
 - **縮小変換**：大きなタイプを小さなタイプに変換。
情報が失われるため、**手動で変換を表記する必要があります**。
- データ型の「サイズ」：大きいデータタイプが小さいタイプを「含む」のです。
 - byte < short < int < long < float < double
 - char < int

- 手動表記の方法：

```
1 int a = 100;
2 char b = (char) a;
```

Try <sup>01011
11010
01011</sup>
Casting.java

Q&A

char 型とエンコーディング (1)

Java の char 型は、整数だけでなく、**文字**も表現できます。実は、char は Character (文字) の略称です。

コンピュータでは、すべてのデータが **2 進数**として保存され、受け渡されます。そのため、文字を直接メモリに保存することはできません。各文字に番号を割り当て、この番号を実際のデータとして保存する必要があります。char はこの番号の値を保存します。

この番号と文字の対応を**エンコーディング**^[encoding]または**文字セット**^[charset]と呼ばれます。言語によって使用する文字コードが異なる場合があります、ファイルが文字化けする原因の一つとなっています。

char 型とエンコーディング (2)

英文字と常用符号は、ほとんどの文字セットで同じようにエンコードされており、「**ASCII**」というエンコード規格に合致しています。例えば、数字の 65 はアルファベットの「A」を表しています。

Try 
ASCII.md

Try 
Char.java

また、異なるエンコーディング間の変換の問題を解決するために、すべての言語の文字を包含するユニバーサルエンコーディングセットがいくつか存在します。

「UTF-8」「UTF-16」などがあります。**すべてのコードを UTF-8 で保存することをお勧めします。**

目次

- 1 変数とデータタイプ
- 2 文字列と配列
- 3 演算子

参照型

- 先ほど紹介した基本データ型以外のタイプを**参照型**[Reference Type]と呼ばれます。基本データ型との主な違いは以下の通りです：
 - 新しい参照型は定義できますが、新しい基本データ型は**定義できません**。
 - 参照型は、**アドレス値（番地）**を参照している。
 - 参照型は**メソッド**を呼び出して特定の操作を行うことができるが、基本型はできません。
 - すべての参照型変数には、特別な値である **null** を格納することができます。
 - 基本データと区別するために、参照型の名前は一般的に**大文字で始まります**。
 - 基本データの格納サイズはデータタイプに依存し、参照型のサイズはオペレーティングシステムに依存します（32 ビットシステムでは 4 Byte、64 ビットシステムでは 8 Byte）。

文字列

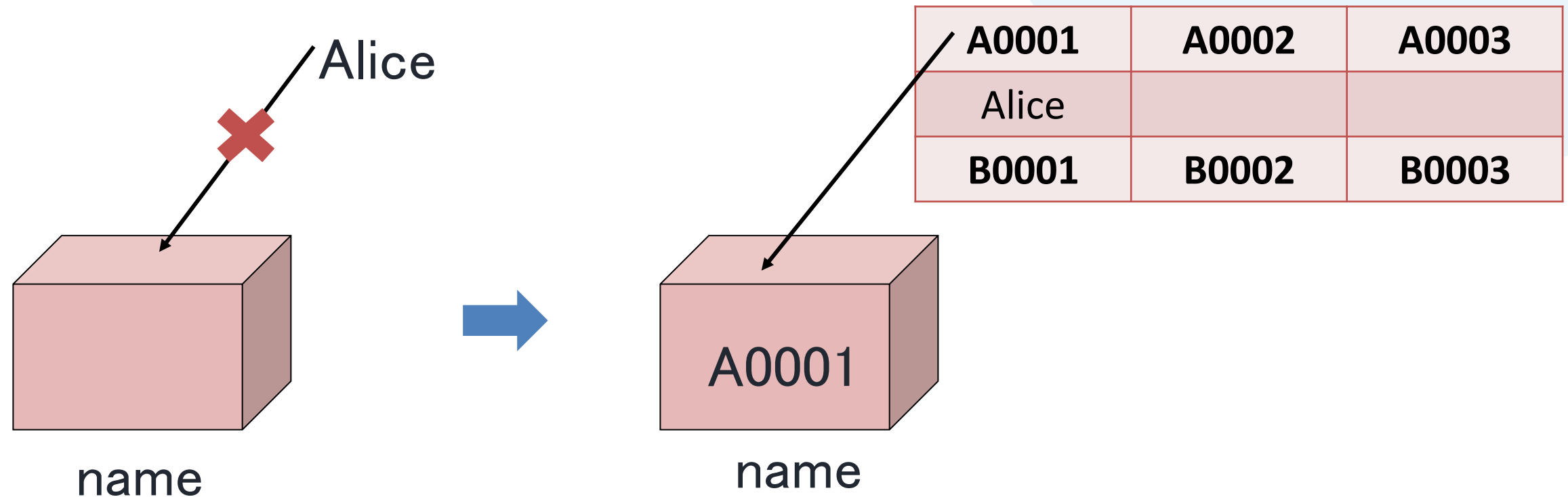
- **文字列**[String]は、テキストを格納するデータタイプで、Javaでは String 型で格納されます。
- 文字列の内容は、二重引用符「**"**」で囲む必要があります。
- 参照型である String には、便利なメソッドが多数付属しています。たとえば、文字列の長さは、次の **length()** メソッドで得ることができます：

```
1 String str = "abcdefghijklm";  
2 System.out.println(str.length()); // => 13
```

- その他のメソッドについては後述します。

参照型の例

```
String name = "Alice";
```



- データ自体は、メモリ上の別の場所に置かれてそのアドレス値（**参照値**）が変数に入る

変数を出力

- この前、System.out.println() メソッドを使って文字列を出力できることを学びました。
- 複数の文字列を表示する場合は、「+」で連結して表示することができます：

```
System.out.println("Hello" + ", " + "World"); // => Hello, World
```

- 文字列と他の種類のデータを「+」で連結することも可能です。連結されるデータは、文字列に変換されます：

```
1 int id = 100;  
2 System.out.println("Hello, " + id + "!"); // => Hello, 100!
```

エスケープ文字

- 特殊文字の中には、コードに直接書くことができないもの（改行など）があり、特殊な方法で表現する必要があります。**エスケープシーケンス**[\[Escape Sequence\]](#)は、このような特殊文字を複数の文字の組み合わせで表現します。Java のエスケープシーケンスは、すべてバックスラッシュ「\」で始まります。このため、バックスラッシュは「**エスケープ文字**」とも呼ばれます。
- 以下は、エスケープが必要な代表的な文字です（なぜエスケープする必要があるのか、考えてみてください）：

文字	書き方	文字	書き方
改行	<code>\n</code>	バックスペース	<code>\b</code>
Tab	<code>\t</code>	「\」	<code>\\</code>
「"」	<code>\"</code>	「'」	<code>\'</code>

配列

- **配列**^[Array]とは、複数の値を 1 つの変数に格納するために使用されるデータ構造で、各値に対して個別の変数を宣言する必要がなくなります。
- Java では、配列を宣言する場合、大括弧「**[]**」で変数のタイプを定義します：

```
int[] arr;
```

- 配列は、**宣言の時のみ**、中括弧「**{}**」で初期化することができます：

```
String[] texts = {"hello", "world", "!"};
```

- 基本データ型かどうかと関係無く、どんなタイプにも対応する配列があります。ただし、同じ配列の中の値はすべて**同じデータタイプ**でなければなりません。

配列の作成手順

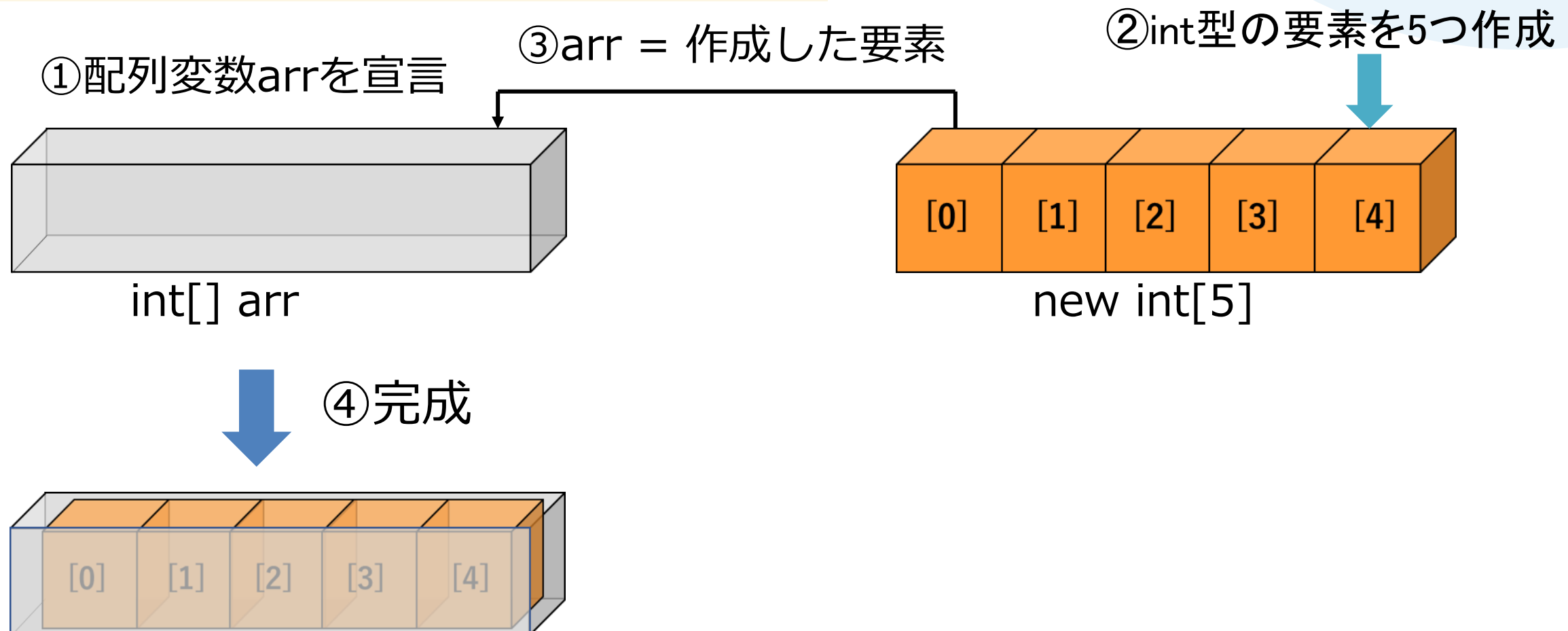
```

1 public class Main {
2
3     public static void main(String[] args) {
4         int[] arr;
5
6         arr = new int[5];
7     }
8 }

```

int型の要素を代入できる配列変数
arrを用意

int型の要素を5つ作成し、arrに代入し、
配列arrの完成



配列の宣言

- 中括弧だけでなく、**new** で配列を宣言することもできます：
 1. 指定サイズの配列を宣言します：

```
texts = new String[3];
```

- 注意：このプログラムでは、配列内のすべての要素が自動的に「デフォルト値」に設定されます。
- 数値のデフォルト値は 0、ブール値は false、参照型は null です。

2. 初期要素が指定された配列を宣言します：

```
texts = new String[] {"a", "b", "c"};
```

- 2 つのシステムで使用するには、このようにします：

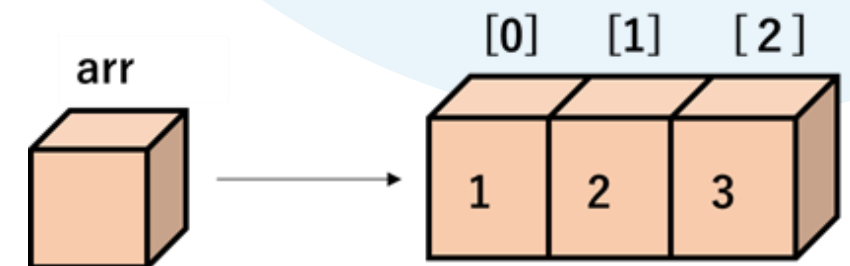
Example

```
int[] arr = new int[2] {1, 2}; // error
```


配列の使い方

- 配列の各データを配列の**要素**[Element]と呼びます。
- 配列の要素は順番に並んでおり、各要素の番号（位置）を**添え字**[Index]と呼びます。配列名の後に大括弧「**[]**」を付ければ、添え字で要素にアクセスできます：

```
1 int[] arr = new int[] {1, 2, 3};
2 arr[1] = 100;
3 System.out.println(arr[1]); // => 100
```



Note



配列の添え字は **0** から始まり、「0」が最初の要素で、「1」は 2 番目の要素です。この特性は、多くのコンピュータ言語で見られます。

配列の使い方

- 配列のサイズは **length** 属性で取得できます: (String の `length()` メソッドとの違いに気づきましたか?) :

```
1 int[] arr = new int[] {1, 2, 3};
2 System.out.println(arr.length); // => 3
```

- 配列の最後の要素の添え字が、配列のサイズとどのような関係になるのか、考えてみましょう。



多次元配列

- 先ほど述べたように、**配列を含む**あらゆるタイプに対応する配列型が存在します。配列の配列を **2次元配列** [2D Array] と呼ばれます。2次元配列の配列は3次元配列です。これらをすべて**多次元配列**と呼びます。多次元配列は画像やネットワークなどの複雑なデータを表現するために使用されます。
- 多次元配列は、通常の配列と同じように宣言します。タイプ（「[]」の前の部分）を「配列型」と書くだけです：

```
int[][] array2D;
```

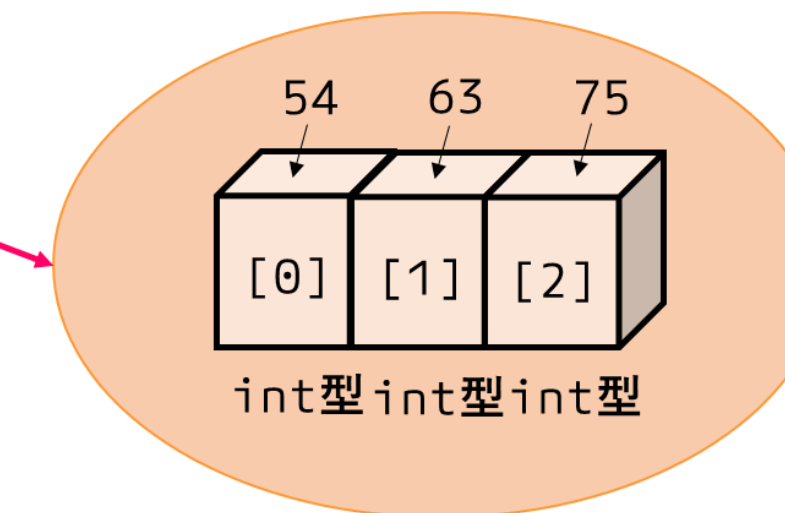
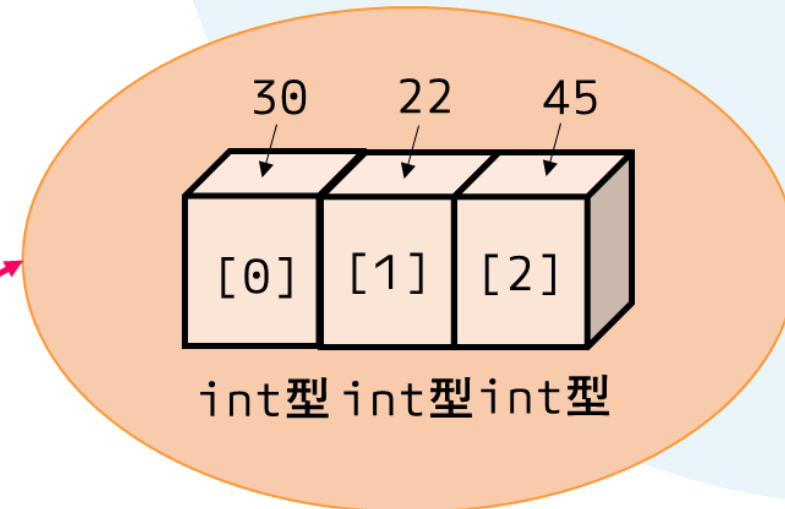
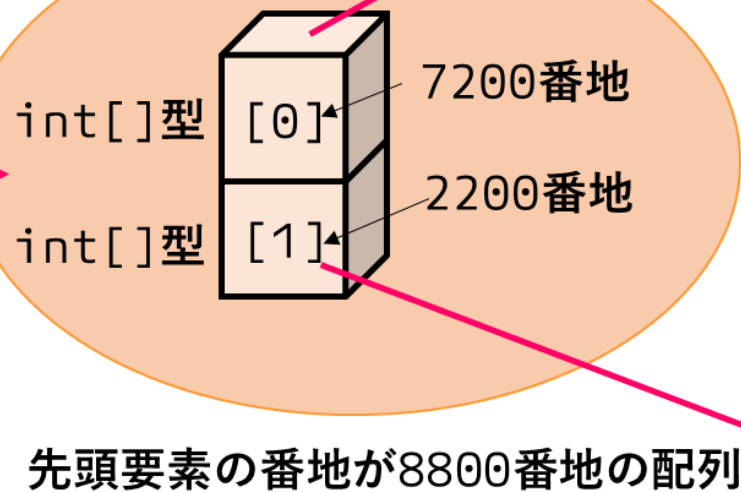
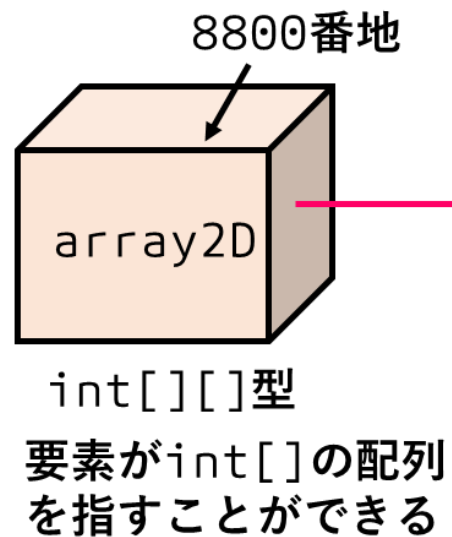
- 1次元配列と同様に、「{ }」で初期化することもできます：

```
int[][] array2D = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```

次へ

メモリ上での2次元配列

- `int[][] array2D = new int[2][3]`



次へ



- new でサイズを指定して配列を宣言する場合、複数の次元のサイズを指定することができます：

```
int[][] array2D = new int[3][3];
```

- また、1次元のみサイズを指定することも可能です（この場合、配列内のすべての配列が **null** になります）：

```
int[][] array2D = new int[3][];
```

- 多次元配列を使用する場合、データを取得するために**複数の「[]」**演算子を使用する必要があります。以下の例では、直前の「2」が「2番目の配列」を意味し、続く「0」が「その配列の0番目の要素」を意味します：

```
System.out.println(array2D[3][0]);
```

Try  MultiArray.java

Q&A

目次

- 1 変数とデータタイプ
- 2 文字列と配列
- 3 演算子

Java 演算子

- **演算子**^[Operator]は、既存のデータに対して演算を行って、新しいデータを作るために使われます。
- Java の演算子は、その機能によって、次のように分けられます：
 - 算術演算子
 - 代入演算子
 - 比較演算子
 - 論理演算子
 - ビット演算子

算術演算子

- **算術演算子**は、算術的な演算を行うために使用されます。

演算子	概要	例
+	加算	1 + 3
-	減算	3.0 - 5.6
*	乗算	6 * -7
/	除算	13 / 5
%	剰余	8 % 3
++	インクリメント（1 だけ増加）	x++
--	デクリメント（1 だけ減少）	y--

Tips 

「++(--)x」は「x++(--)」と書くこともできます。

比較演算子

- **比較演算子**は、2 つの値を比較するために使用されます：

演算子	概要	例
>	大なり	<code>a > 1</code>
<	小なり	<code>b < -7</code>
>=	大なりイコール	<code>c >= 6.6</code>
<=	小なりイコール	<code>d <= 0</code>
==	イコール	<code>x == y</code>
!=	イコールではない	<code>a != b</code>

- 2 つの値は等しいかどうかを判断するには、「==」を使ってください！「=」は代入コマンドです。

論理演算子

- 一つの比較演算子では計算できないような複雑な条件もあります。例えば、「x と y のどちらかが 3 より大きい」、「x と y がともに奇数である」など。この場合、**論理演算子**を使ってブール式を構成する必要があります。
- 論理演算子は、変数や判断条件を論理的に結合するために使用されます：

演算子	概要	例
&&	… かつ[AND] …	<code>a > b && c > d</code>
	… または[OR] …	<code>k > 10 k <= 0</code>
!	… ではない[NOT]	<code>!(a == 0 && c == 0)</code>

ブール式

- **ブール式**^[Boolean Expression]とは、いくつかの論理演算の組み合わせで、ブール値を返す式です。

Example



「x と y がともに 3 より大きい」を表現するには、次のように書けばいい：

```
x > 3 && y > 3
```

- 練習問題：次のコードを実行した結果は何ですか？

```
1 int x = 1;
2 int y = 2;
3 System.out.println(x < y || x > y + 1 && x < 0);
```

- 実際に実行してみてください。結果は予想と一致しますか？

優先順位

- 数学で乗算と除算が常に加算と減算に優先するように、Javaの演算子にも**優先順位**^[Precedence]があります。優先順位の高い演算子が先に計算されます。

Tips

優先順位を忘れがち？小括弧「**()**」を使えば間違いがないでしょう。

種類	演算子
Postfix	a++ a--
Unary	++a --a -a !a
multiplicative	* /
additive	+ -
relational	> >= < <=
equality	== !=
logical AND	&&
logical OR	
assignment	= += -= など

代入演算子

- **代入演算子**は、変数に値を代入するために使われます。

演算子	概要	例
=	代入	a = 3
+=	増加	b += 5
-=	減少	c -= 10
*=	乗算して代入	d *= 2
/=	除算して代入	e /= 3
%=	剰余を計算して代入	f %= 4

Try  Operator.java

Q&A

まとめ

Sum Up



1. データタイプ :

- ① 基本データ型 : 整数、小数、ブール、char。
- ② 参照型 : 文字列 (String)、配列 (Array)。
- ③ キャスト。

2. 演算子 :

- ① 算術演算子。
- ② 比較演算子。
- ③ 論理演算子。
- ④ 代入演算子。
- ⑤ 演算子の優先順位。



Light in Your Career.

THANK YOU!