

第2章

配列



配列とは

同じデータ型の値をまとめて扱う場合に使用する。

例)

intの配列

[1, 3, 5, 10]

Stringの配列

["ABC", "XYZ", "123"]

宣言したデータ型と異なる値は設定できない。

`int intArray[] = new int[]{1, 2}` ... int型しか格納できない配列

`String stringArray[] new String[]{"ABC", "XYZ"}` ... String型しか格納できない配列

配列とは

配列は『参照型』となる。

Stringのような元々参照型の配列だけではなく、
intのようなプリミティブ型の配列も参照型のオブジェクトとなる。

オブジェクト内には格納値以外の情報も保持しているので、
データの個数なども簡単に取得できる。

```
jshell> int aaa[] = new int[]{1,2}
aaa ==> int[2] { 1, 2 }

jshell> aaa.length
$7 ==> 2
```

配列の作成

データ型[] 配列名 = new データ型 [要素数]

or

データ型 配列名[] = new データ型 [要素数]

おぼえよう！

添え字(インデックス)

- ・添え字はゼロから始まる。
- ・初期値の設定と要素数取得の例。

```
jshell> int aaa[] = new int[]{1,2}
aaa ==> int[2] { 1, 2 }

jshell> aaa.length
$7 ==> 2
```

- ・要素の確保だけ行って値を設定しない場合は、型に応じた初期値が設定される。
- ・要素以上のインデックスを指定すると、実行時エラーが発生する。

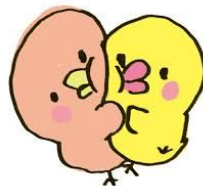
```
jshell> aaa[0]
$8 ==> 1

jshell> aaa[1]
$9 ==> 2

jshell> aaa[2]
| java.lang.ArrayIndexOutOfBoundsException thrown: 2
|   at (#10:1)

jshell> |
```

今後頻繁に見ることになるので、仲良くなろう



多次元配列

配列の中に、さらに配列を格納させることができる。

```
int[][] array = new int[3][4];  
array[0][0] = 100;  
array[0][3] = 200;
```

0	100	0	0	200
1	0	0	0	0
2	0	0	0	0

例は2次元の配列だけど、3次元でも4次元でも作成可能。
→意味わからなくなるから、基本的にはやらない。

多次元配列

array[0][3] の読み解き方。

1. まず、array[0]を取得する。

100	0	0	200
0	0	0	0
0	0	0	0

2. シンプルな配列になったので、あとは 4 番目の要素にアクセスするだけ。

100	0	0	200
-----	---	---	-----

多次元配列

コードにするとかななカンジ。

```
jshell> int[][] array = new int[3][4]
array ==> int[3][] { int[4] { 0, 0, 0, 0 }, int[4] { 0, 0, 0, 0 }, int[4] { 0, 0, 0, 0 } }

jshell> array[0][0] = 100
$12 ==> 100

jshell> array[0][3] = 200
$13 ==> 200

jshell> int[] array_tmp = array[0]
array_tmp ==> int[4] { 100, 0, 0, 200 }

jshell> array_tmp[3]
$15 ==> 200
```

ここの部分だけに着目した変数を作成している。
型宣言も1次元配列であることに注目。

多次元配列

先の例からわかることは、多次元配列とは配列の中に独立した配列が格納されているということ。

なので、以下のようなサイズの配列も作成可能。

配列とArrayListの違い

配列を使うと複数のデータを1つにまとめて扱うことができるので、非常に便利。

ただし、色々扱いが難しい・・・

そこで**ArrayList**です。

ArrayListはjavaのAPIとして提供されている。

ArrayListのように、複数のデータを1つにまとめて扱うことのできるクラスは他にも提供されている。

これらをまとめて、『コレクションフレームワーク』と呼ぶ。

- HashMap
 - HashSet
 - ArrayList
- などなど

配列とArrayListの違い

ArrayList.java (~/Desktop/src/java/util) - VIM

ArrayList.java (~/Desktop/src/java

D/s/j/u/ArrayList.java

```
105 MissingFormatException.
106 MissingFormatWidthExcepti
107 MissingResourceException.
108 NavigableMap.java
109 NavigableSet.java
110 NoSuchElementException.ja
111 Objects.java
112 Observable.java
113 Observer.java
114 Optional.java
115 OptionalDouble.java
116 OptionalInt.java
117 OptionalLong.java
118 package.html
119 PrimitiveIterator.java
120 PriorityQueue.java
121 Properties.java
122 PropertyPermission.java
123 PropertyResourceBundle.ja
124 Queue.java
125 Random.java
126 RandomAccess.java
127 RegularEnumSet.java
128 ResourceBundle.java
129 Scanner.java
130 ServiceConfigurationError
131 ServiceLoader.java
132 Set.java
133 SimpleTimeZone.java
134 SortedMap.java
135 SortedSet.java
136

105 public class ArrayList<E> extends AbstractList<E>
106     implements List<E>, RandomAccess, Cloneable, java.io.Serializable
107 {
108     private static final long serialVersionUID = 8683452581122892189L;
109
110     /**
111      * Default initial capacity.
112      */
113     private static final int DEFAULT_CAPACITY = 10;
114
115     /**
116      * Shared empty array instance used for empty instances.
117      */
118     private static final Object[] EMPTY_ELEMENTDATA = {};
119
120     /**
121      * Shared empty array instance used for default sized empty instances. We
122      * distinguish this from EMPTY_ELEMENTDATA to know how much to inflate when
123      * first element is added.
124      */
125     private static final Object[] DEFAULTCAPACITY_EMPTY_ELEMENTDATA = {};
126
127     /**
128      * The array buffer into which the elements
129      * The capacity of the ArrayList is the leng
130      * empty ArrayList with elementData == DEF
131      * will be expanded to DEFAULT_CAPACITY
132      */
133     transient Object[] elementData; // non-private to simplify nested class access
```

ArrayListの実体はObject型の配列。
ArrayListは配列を扱いやすくするための
機能を提供してくれている

java.util.ArrayListクラスの利用

ひたすら使って覚えよう！

【注意点】

ArrayListはデフォルトで要素を 10個保持しようとする。

必ず要素数が 10個以下になる場合は、要素数を指定することでメモリの節約ができる。

<>に指定可能なデータ型

<E>のように<>で囲って型を指定する機能をジェネリックスと呼ぶ。

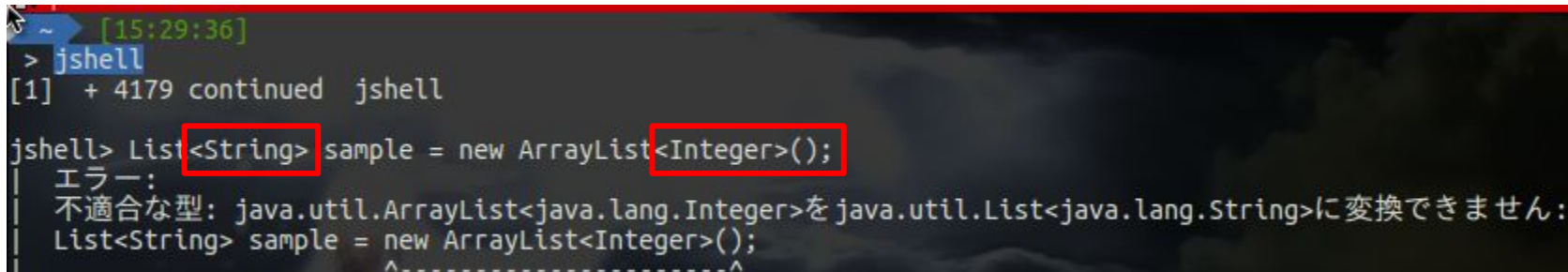
```
List<String> sample = new ArrayList<String>();  
List<Integer> sample = new ArrayList<Integer>();
```

ジェネリックスはめっちゃ奥深い機能です。
続きはGoldで。

上記のように型を指定することで、Stringしか格納できないリスト、Integerしか格納できないリストを用意することができる。

<>に指定する型は参照型のみで、プリミティブな型は指定できない。

左右の型が合っていないとコンパイルエラーとなる。



```
[15:29:36]  
> jshell  
[1] + 4179 continued jshell  
  
jshell> List<String> sample = new ArrayList<Integer>();  
エラー:  
不適合な型: java.util.ArrayList<java.lang.Integer>をjava.util.List<java.lang.String>に変換できません:  
List<String> sample = new ArrayList<Integer>();  
          ^-----^
```

ジェネリックスを使用しないコード

ジェネリックスを使用せずに ArrayList を使用することもできる。

```
jshell> List sample = new ArrayList()  
sample ==> []  
  
jshell> |
```

この場合、格納される値の型が指定されない。
これってつまり、以下と同じ意味になる。

```
jshell> List<Object> sample = new ArrayList<Object>()  
sample ==> []
```

ジェネリクスを使用しないコード

ジェネリクスを使用しない例

```
jshell> List sample1 = new ArrayList()
sample1 ==> []

jshell> sample1.add(new Integer(10))
警告:
raw型java.util.Listのメンバーとしてのadd(E)への無検査呼出しです
sample1.add(new Integer(10))
^-----^
$20 ==> true

jshell> sample1.add("ABC")
警告:
raw型java.util.Listのメンバーとしてのadd(E)への無検査呼出しです
sample1.add("ABC")
^-----^
$21 ==> true

jshell> sample1
sample1 ==> [10, ABC]
```

どんな型の値が入っているかわからないので、実行時にちゃんとチェックしないとキャストエラーが頻発する。

ジェネリクスを使用しないコード

ジェネリクスを使用する例

```
jshell> List<Integer> sample2 = new ArrayList<Integer>()  
sample2 ==> []
```

```
jshell> sample2.add(new Integer(10))  
$24 ==> true
```

```
jshell> sample2.add("ABC")
```

型が不正だとコンパイルエラーになるので、
キャストエラーの心配がない。
←コンパイラが型チェックをしている

エラー:

addに適切なメソッドが見つかりません(java.lang.String)

メソッド java.util.Collection.add(java.lang.Integer)は使用できません

(引数の不一致: java.lang.Stringをjava.lang.Integerに変換できません:)

メソッド java.util.List.add(java.lang.Integer)は使用できません

(引数の不一致: java.lang.Stringをjava.lang.Integerに変換できません:)

```
sample2.add("ABC")
```

```
^-----^
```


ダイヤモンド演算子の利用

左にも右にも<Integer>と書くのがめんどくさいので、省略することも可能。

```
yshell> List<Integer> sample2 = new ArrayList<>()  
sample2 ==> []
```

類似の概念に『型推論』というものもあるけど、この機能は型推論とはちょっと違う。

コマンドライン引数の利用

javaの実行時に渡されたパラメータは String型の配列で渡される。

人間は「数字」だとわかるけど、このような場合でもパラメータは必ず String型で渡される。

```
[15:57:21]  
> java sample 10 200 3000
```

ここに渡される

```
sample.java+  
1 public sample {  
2     public static void main(String[] args) {  
3         sample.java  
4  
5     }  
6 }
```