

プログラミング基礎 I

第5回 配列 発展・応用

九州産業大学 理工学部

pk@is.kyusan-u.ac.jp

内容

- 参照型配列を扱う上での注意
 - shallow copy(シャローコピー)、Deep copy(ディープコピー)
- 指定文字列で文字列を分割
 - String.splitメソッド
- 文字列を文字配列に変換
 - String.toCharArrayメソッド

参照型変数を扱う場合の注意

```
int [] arrayA = {1, 2, 3, 4, 5};  
int [] arrayB = arrayA; // arrayAにarrayBをコピー  
System.out.print("arrayAを出力 : ");  
for(int i: arrayA) {  
    System.out.print(i);  
}  
System.out.println();  
  
System.out.print("arrayBを出力 : ");  
for(int i: arrayB) {  
    System.out.print(i);  
}  
System.out.println();  
  
arrayA[0] = 9999; // arrayAの要素を変更(ArrayBは変更していない)  
System.out.print("arrayAを出力 : ");  
for(int i: arrayA) {  
    System.out.print(i);  
}  
System.out.println();  
  
System.out.print("arrayBを出力 : ");  
for(int i: arrayB) {  
    System.out.print(i);  
}  
System.out.println();
```

arrayAを出力 : 12345
arrayBを出力 : 12345
arrayAを出力 : 99992345
arrayBを出力 : 99992345

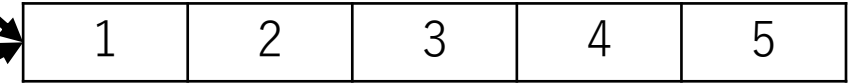
arrayBを変更した覚えがないのに、
arrayBも変わってしまった・・・

```
int [] arrayA = {1, 2, 3, 4, 5}; arrayA
```

変数名は別でも、実体は同じデータ

```
int [] arrayB = arrayA; arrayB
```

arrayAと同じ場所を参照



配列などの参照型の変数を = で代入すると、
データの実体があるアドレスへの参照が代入される

同じところを指しているので、
どちらかを変更するともう一方も変更されてしまう

シャローコピー（浅いコピー）

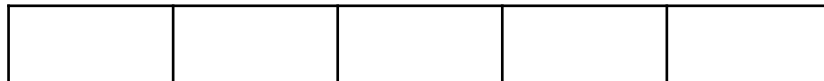
参照している先のコピー（実体は同じ）、見せかけの複製を作る 4

```
int [] arrayA = {1, 2, 3, 4, 5};
```

```
int [] arrayA = arrayB;
```

arrayAと同じ場所を参照

arrayA ☐

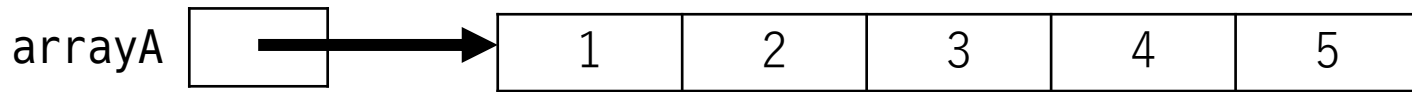


arrayB ☐

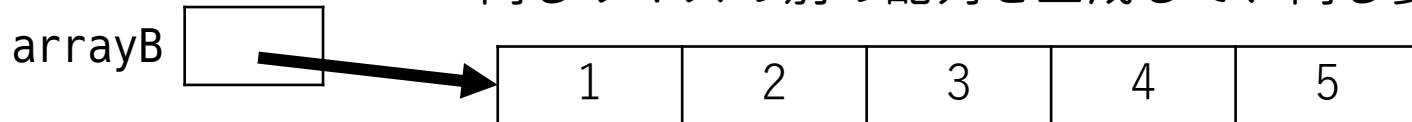


回避方法

```
int [] arrayA = {1, 2, 3, 4, 5};  
int [] arrayB = new int[arrayA.length]; // arrayAと同じサイズの配列を新しく生成  
for(int i = 0; i < arrayB.length; i++) {  
    arrayB[i] = arrayA[i]; //  
}
```



同じサイズの別の配列を生成して、同じ要素を代入



配列の各要素は原始型(int型)

ディープコピー（深いコピー）

データの参照だけでなく、データの実体までのコピー
中身も完全に複製するコピー

参照型のデータコピー

- 配列やクラス型のオブジェクトの複製を作る際の注意
 - 単に代入するだけだと、データの参照先が複製される
 - シャローコピー（浅いコピー）
 - 同じ場所を参照しているので、変更すると元のオブジェクトも変更される
- 別々のデータとして複製したい場合は、新たなオブジェクトを生成して、原始型のデータをコピーする
 - ディープコピー（深いコピー）

指定文字列で文字列を分割

- **String.split**メソッド

- 特定の文字列で区切られた文字列を、部分文字列に分割し、**String型の配列**として返すメソッド

- ```
String text = "a,b,c,d,e,f,g";
String [] splitText = text.split(",");
// textを , 区切りで分割して String型配列のsplitTextに代入
```

|           |   |   |   |   |   |   |   |
|-----------|---|---|---|---|---|---|---|
| splitText | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|           | a | b | c | d | e | f | g |

# 文字列を分割( String.splitメソッド )

- カンマ区切りで入力されるデータを処理する例

- 1,2,3,4 の文字列を、

|   |   |   |   |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
|---|---|---|---|

の配列に

```
String str4 = "1,2,3,4";
String [] spStr3 = str4.split(","); //str4をカンマで分割して配列に
for(String s : spStr3) {
 System.out.print(s + " ");
}
```

- csvファイルなど、カンマ区切りで入力されるデータを処理する時などによく利用する

# 分割した文字列を整数配列に変換

- 同じサイズの整数型の配列を用意、  
文字列を整数値に変換すると、整数型配列として扱える

```
String str4 = "1,2,3,4";
String [] spStr3 = str4.split(",");
int [] intStr3 = new int[spStr3.length];
for(int i = 0; i < spStr3.length; i++) {
 intStr3[i] = Integer.parseInt(spStr3[i]);
}
```

- 文字列が整数値に変換出来る場合に限る

# char型

- char型：単一の文字を格納出来る変数型
  - 1バイトの文字列を表現できる
  - 文字は、シングルクォート( ' )で囲む
  - Unicode文字を表現できる

```
char c1 = 'A'; //英数字
char c2 = 65; // char型に整数値を代入
System.out.println(c1); // A
System.out.println(c2); // A
```

- 65 を代入して A と表示される
  - 'A' は ASCIIコード表で 0x41 つまり、10進数で65

# char型の便利な使い方（例）

- A ~ Z を出力する

```
char c1 = 'A'; //英数字
for(int i = 0; i < 26; i++) {
 System.out.print(c1 + ", ");
 c1 += 1;
}
```

- ASCIIコードの並びを利用すると、  
アルファベットを繰り返し1回で出力出来る

# String型をchar型配列に変換

- String型(文字列)をchar型の配列に変換する
  - String.toCharArrayメソッドでchar型配列に変換

```
String str5 = "convert Test";
char [] cArray = str5.toCharArray();
```

|   |   |   |   |   |   |   |  |   |   |   |   |
|---|---|---|---|---|---|---|--|---|---|---|---|
| c | o | n | v | e | r | t |  | T | e | s | t |
|---|---|---|---|---|---|---|--|---|---|---|---|

char型配列に変換することで、  
文字列の処理を配列の処理としても行うことが出来るようになる