Java プログラミング入門

— Java プログラミングの基礎：配列 —

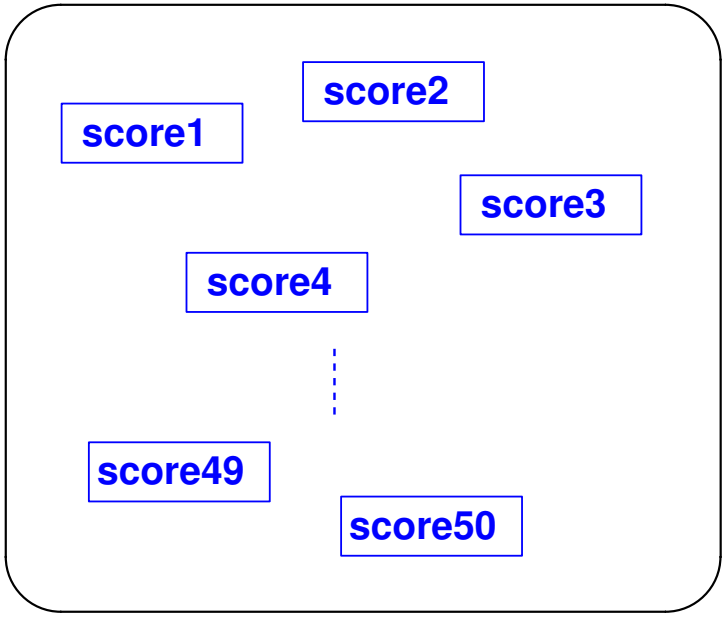
早稲田大学

問題

例題

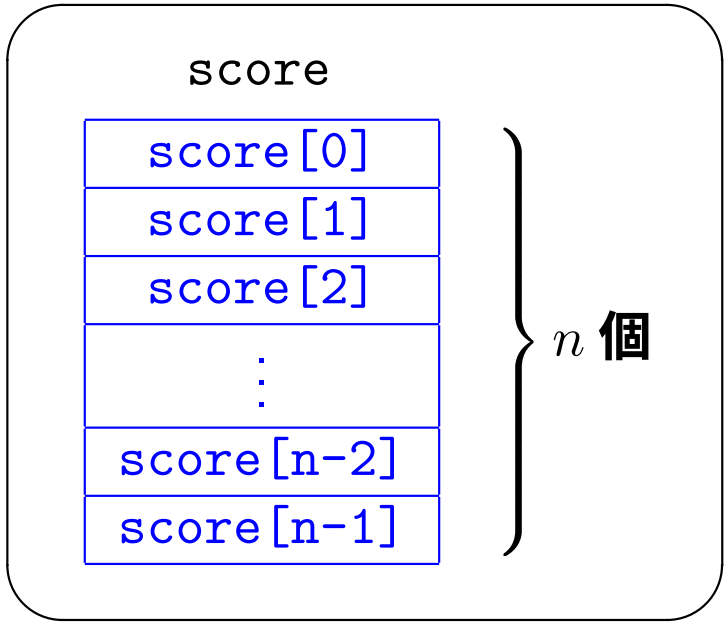
ある科目のテストについて，50 人分の点数のデータが与えられている．このとき，平均点を表示しなさい．

- 50 人分のデータを扱うため，50 個の変数が必要
- 50 個の変数を宣言したり管理するのは大変．また，平均点などを求める処理もややこしい
- このとき，50 人分のデータは “テストの点数” という共通の性質を持った変数として扱うことができる
- プログラミングでは**配列**と呼ばれるものを利用すると，多数の変数を効率良く扱うことができる



配列

- 同じ型の変数を複数個まとめて扱うことができるデータの構造
- 変数と同様に，利用するには宣言をする必要がある（方法は後ほど説明）
- 変数と同様に，型がある
- 配列を使用すると，右図のように n 個の変数を 1 つの変数名で利用できるようになる



配列の宣言（1）

例：“int 型の配列 data” を宣言する

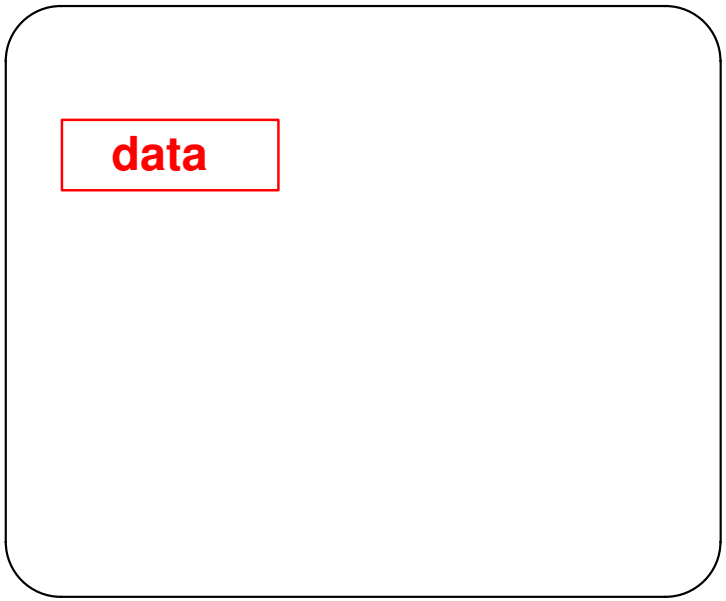
1. **変数名の宣言**

   `int[] data;` — **方法 1**

   または

   `int data[];` — **方法 2**

   - **どちらの方法でも構わないが，この講義では方法 1 で統一する**



配列の宣言（1）

例：“int 型の配列 data” を宣言する．

1. **変数名の宣言**

   `int[] data;` — **方法 1**

   または

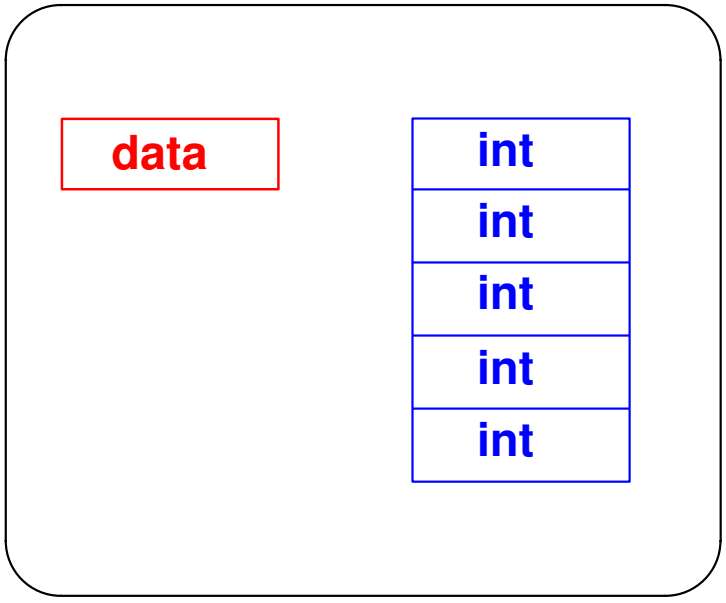
   `int data[];` — **方法 2**

   - **どちらの方法でも構わないが，この講義では方法 1 で統一する**

2. **配列の実体の作成**

   `new int[5]`

   - `int` **型 5 個分の領域が確保される**



配列の宣言（1）

例：“int 型の配列 data” を宣言する．

1. **変数名の宣言**

   `int[] data;` — **方法 1**

   または

   `int data[];` — **方法 2**

   - **どちらの方法でも構わないが，この講義では方法 1 で統一する．**

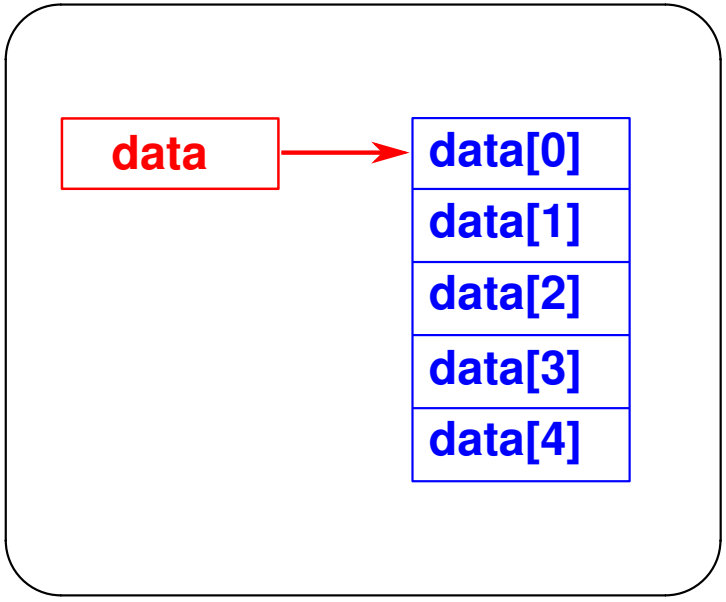
2. **配列の実体の作成**

   `new int[5]`

   - **int 型 5 個分の領域が確保される．**

   `data = new int[5];`

   - **確保した領域を変数 data に代入することで，data が配列の実体を持つ．**



配列の宣言（2）

```
int[] data;
data = new int[5];
```

- 上の 2 つにより，“int 型の配列 data” が宣言できる
- このとき，data は 5 つの**要素**で構成される配列となる
- data の要素は，

  data[0], data[1], data[2], data[3], data[4]

  の 5 つ．各要素を int 型の変数と同様に扱うことができる
- 配列名につづく [ ] 内を**添字**と呼ぶ．添字は **0 から始まる**ことに注意!!

配列の宣言（3）

```
int[] data = new int[5];
```

- 上の 2 つの手続きをまとめて記述することができる
- こちらの方が，より簡潔な記述である

配列の利用（1）

```
int[] data = new int[5];

data[0] = 10;
data[1] = data[0]*3;
```

- **配列を宣言することで，配列の各要素を変数と同様に扱うことができる**
- **上の場合，**`data[0]`～`data[4]`**の** 5 **つの要素が変数となる**

配列の利用（2）

```
public class SampleArray {
  public static void main(String[] args) {
    int[] data = new int[5];
    for (int i=0; i<5; i++) {
      data[i] = 100 - i*10;
      System.out.println(data[i]);
    }
  }
}
```

```
data[0] = 100
data[1] = 90
data[2] = 80
data[3] = 70
data[4] = 60
```

- 配列の添字には変数を使うことができる
- 多くの場合，配列の処理はループと組み合わせて使用する

配列の利用（3）

配列の要素数を超えた場合 (エラー)

- 配列の要素数を超えた範囲にアクセスしようとすると，エラーとなる
- このとき，コンパイルはできるが実行時にエラーとなる

```
int i;
int[] data = new int[5];
for (i=0; i<=5; i++) {
  data[i] = 100 - i*10;
  System.out.println(data[i]);
}
```

- 上の例では，data[5] を参照しようとするが，存在しない
- コンパイルはできるが，実行時に以下のようなエラーが出る

```
>> Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
```

配列の利用（4）

配列の長さ

- Java では，配列の長さ（要素数）の情報が保存されている
- 配列の長さは “配列名.length” で参照できる

```
int i;
int[] data = new int[5];
for (i=0; i<data.length; i++) {
  data[i] = 100 - i*10;
  System.out.println(data[i]);
}
```

- data.length の値は 5.

配列の初期化（1）

```
int[] data = {100, 90, 80, 70, 60};
```

- 変数と同様に，配列も**初期化**をすることができる
- 初期化の方法は，配列名の宣言のあとに，`{ }` で囲んだ中に値を列挙することで，それらの値が配列の先頭要素から順に格納される
- 列挙された個数から自動的に配列の要素数が決定されるため，`new` を用いて要素数を記述する必要はない
- 上の場合，`data[0]` から `data[4]` まで順に 100, 90, 80, 70, 60 が格納される

```
int[] data;
data = {100, 90, 80, 70, 60};  — ×
```

- 上の場合，初期化にならないのでエラー

配列の初期化（1）

```
int[] data = {100, 90, 80, 70, 60};
```

- 変数と同様に，配列も**初期化**をすることができる
- 初期化の方法は，配列名の宣言のあとに，`{ }` で囲んだ中に値を列挙することで，それらの値が配列の先頭要素から順に格納される
- 列挙された個数から自動的に配列の要素数が決定されるため，`new` を用いて要素数を記述する必要はない
- 上の場合，`data[0]` から `data[4]` まで順に 100, 90, 80, 70, 60 が格納される

```
int[] data;
data = {100, 90, 80, 70, 60};   — ×
```

- 上の場合，初期化にならないのでエラー

配列の初期化（2）

配列の初期値

- Java の配列では，配列の宣言を行うと，各要素には宣言した型に応じた初期値が格納される．
- `int` 型配列の場合は 0 が，`double` 型配列の場合は 0.0 が各要素に格納される．

変数の場合

```
int data;                    — 変数宣言のみ．初期化は行っていない．
System.out.println(data);    — エラー
```

配列の場合

```
int i;
int[] data = new int[5];
for (i=0; i<5; i++) System.out.println(data[i]);
```

```
0
0
0
0
0
```

配列の初期化（2）

配列の初期値

- Java の配列では，配列の宣言を行うと，各要素には宣言した型に応じた初期値が格納される．
- int 型配列の場合は 0 が，double 型配列の場合は 0.0 が各要素に格納される．

変数の場合

```
int data;                          — 変数宣言のみ．初期化は行っていない．
System.out.println(data);          — エラー
```

配列の場合

```
int i;                                                    0
int[] data = new int[5];                                  0
for (i=0; i<5; i++) System.out.println(data[i]);          0
                                                          0
                                                          0
```

配列の初期化（2）

配列の初期値

- Java の配列では，配列の宣言を行うと，各要素には宣言した型に応じた初期値が格納される．
- int 型配列の場合は 0 が，double 型配列の場合は 0.0 が各要素に格納される．

変数の場合

```
int data;                          — 変数宣言のみ．初期化は行っていない．
System.out.println(data);          — エラー
```

配列の場合

```
int i;
int[] data = new int[5];
for (i=0; i<5; i++) System.out.println(data[i]);
```

```
0
0
0
0
0
```

例題のプログラム

例題

ある科目のテストについて，50 人分の点数のデータが与えられている．このとき，平均点を表示しなさい．

この例題について，プログラム中では 50 人分の点数のデータを与える必要がある．

ここでは `i` 番の学生について次の式で計算される点数を与えることにする：

```
(i*83 + 15) % 101
```

final 変数（1）

- プログラムの中で**定数**として扱われる変数は，final **変数**として宣言することができる
- 変数宣言の前に “final” という装飾子をつける
- final 変数は初期化するべき
- final 変数は値を変更することができない
- final 変数の名前は，通常の変数と区別するために，すべて大文字でつけることが推奨されている

```
final int SIZE = 50;
SIZE = 100;                    — ×
```

- int 型変数 SIZE を final 変数として宣言し，50 で初期化
- これにより，変数 SIZE の内容は書き換えることができない

final 変数（1）

- プログラムの中で**定数**として扱われる変数は，final **変数**として宣言することができる
- 変数宣言の前に“final”という装飾子をつける
- final 変数は初期化するべき
- final 変数は値を変更することができない
- final 変数の名前は，通常の変数と区別するために，すべて大文字でつけることが推奨されている

```
final int SIZE = 50;
SIZE = 100;                    — ×
```

- int 型変数 SIZE を final 変数として宣言し，50 で初期化
- これにより，変数 SIZE の内容は書き換えることができない

final 変数（2）

“例題” のプログラムは....

- **人数を表す “50” は定数と見なせるので，これを final 変数で宣言**
- **各学生の点数を (i*83 + 15) % 101 で計算**

50 人分のデータを与えるところまでの部分：

```
int i;
final int SIZE = 50;
int[] score = new int[SIZE];

for (i=0; i<SIZE; i++) {
  score[i] = (i*83 + 15) % 101;
}
```

キャスト演算子（1）

“例題” の平均点を求める部分は....

- 平均点は （全学生の点数の合計）/（人数） で計算
- 合計を表す変数 sum を int 型で，平均を表す変数 ave を double 型で宣言

```
sum = 0;
for (i=0; i<SIZE; i++) {
   sum += score[i];
}
ave = sum / SIZE;
```

- sum と SIZE はともに int 型
- “sum / SIZE” は小数点以下が切り捨てられて int 型として計算される
- このような場合には，キャスト演算子が便利

キャスト演算子（2）

キャスト演算子

- データの型を強制的に変換する機能
- 記述の仕方：　（型）式
- これにより，式の部分で与えられるデータの型が，その前に記述された “型” に変換される

“例題” の平均点を求める部分は....

```
sum = 0;
for (i=0; i<SIZE; i++) {
  sum += score[i];
}
ave = (double) sum / SIZE;
```

キャスト演算子（2）

キャスト演算子

- データの型を強制的に変換する機能
- 記述の仕方：　（型）式
- これにより，式の部分で与えられるデータの型が，その前に記述された “型” に変換される

“例題” の平均点を求める部分は....

```
sum = 0;
for (i=0; i<SIZE; i++) {
  sum += score[i];
}
ave = (double) sum / SIZE;
```

例題のプログラム例

Ave.java

```
public class Ave{
  public static void main(String[] args) {
    int i,sum;
    double ave;
    final int SIZE = 50;
    int[] score = new int[SIZE];

    for(i=0;i<SIZE;i++){
      score[i] = (i*83 + 15) % 101;
    }

    sum = 0;
    for(i=0;i<SIZE;i++){
      sum += score[i];
    }

    ave = (double) sum / SIZE;
    System.out.println("ave : " + ave);
  }
}
```

2 次元配列 (1)

例：data という名前の int 型の 2 次元配列を宣言する．

```
int[][] data;
data = new int[4][3];
```

data[0][0]	data[0][1]	data[0][2]
data[1][0]	data[1][1]	data[1][2]
data[2][0]	data[2][1]	data[2][2]
data[3][0]	data[3][1]	data[3][2]

- 3 つの要素からなる配列が縦に 4 つ並んでいるといったイメージ
- 各要素 data[i][j] が int 型の変数として使用できる

2 次元配列 (1)

例：data という名前の int 型の 2 次元配列を宣言する．

```
int[][] data;
data = new int[4][3];
```

data[0][0]	data[0][1]	data[0][2]
data[1][0]	data[1][1]	data[1][2]
data[2][0]	data[2][1]	data[2][2]
data[3][0]	data[3][1]	data[3][2]

- 3 つの要素からなる配列が縦に 4 つ並んでいるといったイメージ
- 各要素 data[i][j] が int 型の変数として使用できる

2 次元配列 (1)

例：data という名前の int 型の 2 次元配列を宣言する．

```
int[][] data;
data = new int[4][3];
```

data[0][0]	data[0][1]	data[0][2]
data[1][0]	data[1][1]	data[1][2]
data[2][0]	data[2][1]	data[2][2]
data[3][0]	data[3][1]	data[3][2]

- 3 つの要素からなる配列が縦に 4 つ並んでいるといったイメージ
- 各要素 data[i][j] が int 型の変数として使用できる

2 次元配列 (2)

```
int[][] data = new int[4][3];
```

- 2 つの手続きをまとめることもできる.