発展プログラミング演習II

3. 配列

今回の内容

- 配列の定義と内部表現
- 配列の使い道

- 教科書 2.5節 (pp. 32〜42)

配列 array

- 順序付けられた要素の集まりを保持
  - 同じ処理を順番に繰り返す場合などに便利
  - 同じ型の変数の集まり
- C言語の配列と似ている
  - 細かくは色々と違いがある!!

例題3.0

- 5人の身長を扱うプログラム（断片）

```
public void printHeight() {
  float height1 = 168.4F;
  float height2 = 156.7F;
  float height3 = 182.5F;
  float height4 = 173.9F;
  float height5 = 170.8F;

  // 全員の身長を表示する
  System.out.println(height1);
  System.out.println(height2);
  System.out.println(height3);
  System.out.println(height4);
  System.out.println(height5);
}
```

このプログラムの問題点

- 効率が悪い
  - 人数が増えたら変数と表示処理を書き足さなければならない
  - コーディングミスにつながりやすい
- 応用が利かない
  - n番目のデータだけ修正するには？
  - 身長順に並び替えるには？
  - データを引数で受け取って表示するには？

改善例

- 配列を使う
  - データの保持
  - カウンタ変数を使ったデータの参照

```
public void printHeightArray() {
  float[] heights = {168.4F, 156.7F, 182.5F, 173.9F, 170.8F};

  // 全員の身長を表示する
  for (int i = 0; i < heights.length; i++) {
    System.out.println(heights[i]);
  }
}
```

配列の宣言

型名[] 配列変数名;

または

型名 配列変数名[];

例

```
int[] intArray;
```

```
Sring stringArray[];
```

配列の作成(1)

- new 演算子を使う方法

配列変数名 = new 型名[要素数];

例

```
int[] intArray;
intArray = new int[10];
```

```
int[] intArray = new int[10];
```

これはダメ

```
int intArray[10];  // コンパイルエラー!!
```

new 演算子

- 新しいオブジェクトまたは配列を作成する
  ＝メモリを確保する

例

```
int[] intArray;                ←この時点では空っぽ
intArray = new int[10];        ←ここでint10個分のメモリが
                                 確保される
```

```
int[] intArray;                ←この時点では空っぽ
int a = intArray[5];           ←エラー!!
```

配列の作成(2)

- 初期化子を使う方法

```
型名[] 配列変数名 = {データ1, データ2, ...};
```

例

```
int[] even = {0, 2, 4, 6, 8, 10};
```

配列の大きさは自動的にデータの個数になる

配列の作成(3)

- new 演算子と初期化子の組み合わせ

```
型名[] 配列変数名;
配列変数名 = new 型名[サイズ] {データ1, データ2, ...};
```

例

```
int[] fibonacchi;
fibonacchi = new int[] {0,1,1,2,3,5,8};
```

配列中の要素の参照

配列変数名[インデックス式]

- インデックス式は int 型の値になる任意の式
- 有効な添字の範囲は 0 ～ サイズ － 1
- （ここまではC言語と同様）

- インデックス式が範囲外の場合は例外 IndexOutOfBoundsException が投げられる（例外については後述）

例題3.1

右のプログラムの空欄を埋めて、0から始まるフィボナッチ数列を20個表示するプログラムを完成させ、コンパイルして実行せよ。

フィボナッチ数列：
$F_0 = 0, F_1 = 1,$
$F_{n+2} = F_n + F_{n+1}$ $(n \geqq 0)$

```
public class Fibonacchi {
  public void run() {
    int[] fibo = ____________;
    fibo[0] = 0;
    fibo[1] = 1;
    for (int i = 0; i < _______; i++) {
      fibo[i + 2] = ____________________________;
    }
    for (int i = 0; i < 20; i++) {
      System.out.println(fibo[i]);
    }
  }

  public static void main(String[] args) {
    Fibonacchi app = new Fibonacchi();
    app.run();
  }
}
```

配列の長さ

- 配列の長さは一旦作成すると変更できない
- 作成時に指定する長さは計算式でもよい
  例

```
int size = 8;
int[] numbers = new int[size*2+1];
```

- 配列の長さを参照できる

配列変数名.length

例題3.2

右のプログラムは先のプログラムを、与えられた引数の個数分フィボナッチ数列を表示するように修正したものである。プログラムを完成させて、実行してみよ。

```
public class Fibonacchi {
  public void run(          ) {
    int[] fibo = new int[num];
    fibo[0] = 0;
    fibo[1] = 1;
    for (int i = 0; i < fibo.          - 2; i++) {
      fibo[i + 2] = fibo[i] + fibo[i + 1];
    }
    for (int i = 0; i < fibo.          ; i++) {
      System.out.println(fibo[i]);
    }
  }

  public static void main(String[] args) {
    Fibonacchi app = new Fibonacchi();
    int num = Integer.parseInt(args[0]);
    app.run(     );
  }
}
```

配列の初期化

- 初期化子で初期化した場合、{}内のリストの値で初期化される
- new で作成した配列は下記の値で初期化される
  - boolean型　false
  - 参照型（オブジェクト型）　null
  - 上記以外　0

オブジェクト型の配列*

- newしただけでは実体がないことに注意

NG

```
String[] fruits = new String[10];
System.out.println(fruits[0]);    // NullPointerExceptionとなる
```

OK

```
String[] fruits = {“orange”, “apple”, “banana”};
System.out.println(fruits[0]);
```

```
String[] fruits = new String[10];
fruits[0] = new String(“orange”);
System.out.println(fruits[0]);
```

nullと配列*

- 配列型変数にはnullを代入可能
  例：　int[] intArray = null;
- 長さ0の配列とは別もの
  例：　int[] intArray = new int[0];
  （積極的に使う意味は無い）

拡張forループ（JDK5より）*

- 配列の全要素に対して処理を行うループ
  インデックス変数を書くのが面倒

```
for (int i = 0; i < array.length; i++) {
  System.out.println(array[i]);
}
```

- →このような書き方ができる
  （配列の全要素に対するループとなる）

```
for (int a : array) {
  System.out.println(a);
}
```

その他の配列に関する注意

- Java言語では、文字列にはString型を使う
  - C言語のようにchar[]を使わない
- 単に要素の集合を扱うのならCollectionを使う方法もある
  - ArrayListなど

例題3.3

- 次の仕様のメソッドを持つクラスArrayCat を作成し、実行して動作を確認せよ。
  - シグネチャは public void run()
  - int型で大きさが10の配列を作成し、3から始まる3の倍数10個で初期化
  - int型で大きさが20の配列を作成し、先頭から10個を2から始まる2の倍数で初期化
  - 後者の配列の11番目以降の要素に、前者の配列の内容をコピー
  - 後者の配列の要素を逆順に表示

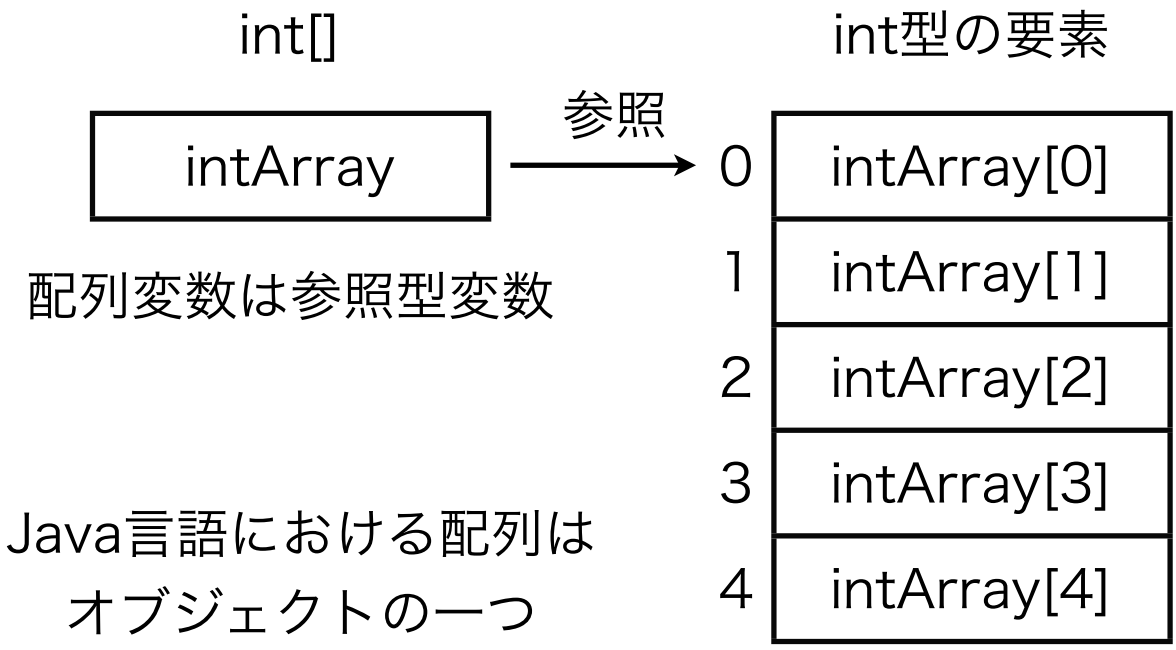
```
public class ArrayCat {
    public void run() {
        // ここを書く
    }

    static public void main(String[] args) {
        ArrayCat app = new ArrayCat();
        app.run();
    }
}
```

課題3.1

- 次の仕様のメソッドを持つクラスArrayCopyを作成し、実行して動作を確認せよ。
  - シグネチャは public void run()
  - int型で大きさが16の配列を作成し、*i*番目の要素の値が2^iとなるように初期化
  - もう一つ、別のint型の配列を作成し、上記の配列の各要素の値を逆順にコピー
  - 後者の配列の要素の値を、奇数個目のみ表示

オブジェクトとしての配列*



配列変数は参照型変数

Java言語における配列は
オブジェクトの一つ

配列への参照の例*

- cf. 教科書 例2.12
- valuesとpは同じ配列の実体を参照するようになる
- 元々pが指していた{3, 2}の配列はいずれ消去される（ガベージコレクション）

```
public class Confusing {
  public void run() {
    int[] values = {1, 2, 3};
    int[] p = {3, 2};
    p = values;
    p[0] = 0;
    System.out.println(values[0]);
  }

  public static void main(String[] args) {
    Confusing c = new Confusing();
    c.run();
  }
}
```

配列変数の値*

- 教科書 例2.13
- 配列変数の値は配列の内容とは無関係
- 参照先のアドレス（＝ポインタ）

```
public class Example0213 {
  public static void main(String[] args) {
    int[] phys = {10, 20};
    int[] phys2;
    phys2 = phys;
    int[] phys3 = {10, 20};

    System.out.println(phys == phys2);
    System.out.println(phys == phys3);
  }
}
```

Java言語は引数は常に値渡し*

- pとiは別々の変数
- qとaは別々の変数だが、同じ配列の実体を指す
- a[0]= 0の代わりに a = new int[]{0, 0, 0}; としたらどうなるか？
- 「オブジェクト型は参照渡し」という理解は誤解しやすいので注意

```
public class Confusing {
  public void run() {
    int p = 1;
    int[] q = {1, 2, 3};
    zero(p, q);
    System.out.println(p);
    System.out.println(q[0]);
  }

  public void zero(int i, int[] a) {
    i = 0;
    a[0] = 0;
  }

  public static void main(String[] args) {
    Confusing c = new Confusing();
    c.run();
  }
}
```

mainメソッドの引数

- public static void main(String[] args)
- argsにはコマンドラインのクラス名以降の引数が文字列として格納される（インデックスに注意）
  - 文字列をint型に変換するにはInteger.parseInt(String)を使う
  - 文字列をdouble型に変換するにはDouble.parseDouble(String)を使う
  - ……（他の型でも同様）

例題3.4

- コマンド行引数で与えられた2つの数の間で四則演算を行うプログラムCalc.javaを書け
  - 2つの数は共に整数とする
  - 引数に使う積の記号は*でなくxを使う（コマンドシェルが*を展開してしまうため）
  - 除算は商と余りを出力するようにせよ
- 実行例：

```
$ java Calc 5 + 3
8
$ java Calc 20 / 3
6...2
```

- 改行せずに表示するにはSystem.out.print()を使うとよい

二次元配列の宣言

型名[ ][ ] 配列変数名;

または

型名 配列変数名[ ][ ];

例

```
int[][] intArray;
```

```
String stringArray[][];
```

二次元配列の作成

```
new 型名[要素数][要素数];
```

または

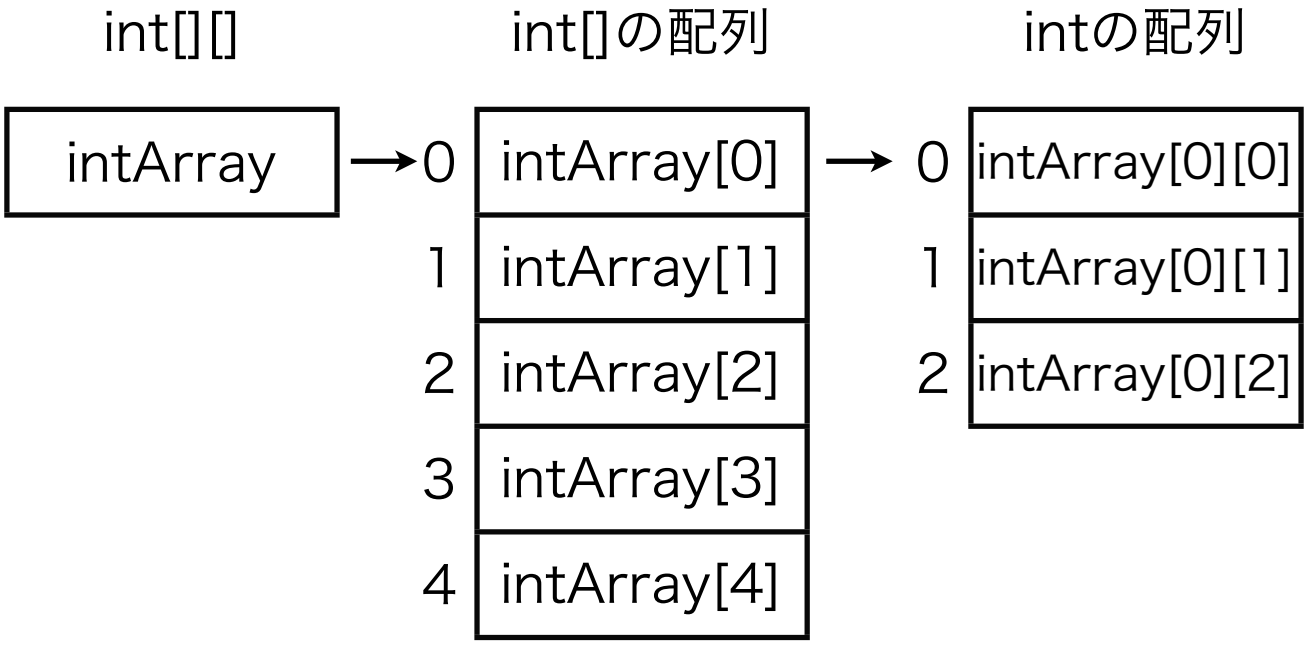
```
{{データ00, データ01, ...},{データ11, データ12, ...}, ...}
```

例

```
int[][] intArray = new int[3][5];
```

```
String stringArray[][] =
                    {{“バナナ”, “りんご”},{“いぬ”,”ねこ”,”さる”}};
```

多次元配列＝配列の配列


int[][]
int[]の配列
intの配列
intArray
0
1
2
3
4
intArray[0]
intArray[1]
intArray[2]
intArray[3]
intArray[4]
0
1
2
intArray[0][0]
intArray[0][1]
intArray[0][2]

例題3.4

- 下のプログラムを実行して動作を確認せよ

```
public class Array2D {
  static public void main(String[] args) {
    int[][] array1 = new int[3][3];
    int[][] array2 = {{1, 2, 3}, {4, 5}, {6}};

    for (int i = 0; i < 3; i++) {
      System.out.println("array1[" + i + "]  " + array1[i].length);
      System.out.println("array2[" + i + "]  " + array2[i].length);
    }

    array1[2][2] = 9;
    array2[2][2] = 9;
  }
}
```

例題3.5

- 3行3列の行列の和を求めるメソッド
  int[][] sum(int[][] a, int[][] b)
  を持つクラスArray3Dを作成せよ

- このメソッドに、$a=\begin{pmatrix}1&2&3\\4&5&6\\7&8&9\end{pmatrix}$, $b=\begin{pmatrix}9&2&3\\8&1&4\\7&6&5\end{pmatrix}$
  を与え結果を出力する処理をmainメソッドに書け

- 配列の初期化、エラー処理は手抜きでよい

例題3.6

- 例題3.5で作成したクラスArray3Dに、
  3行3列の行列の積を求めるメソッド
  int[][] multiple(int[][] a, int[][] b)
  を追加せよ
- このメソッドに、$a=\begin{pmatrix}1&2&3\\4&5&6\\7&8&9\end{pmatrix}$, $b=\begin{pmatrix}9&8&7\\6&5&4\\3&2&1\end{pmatrix}$
  を与え結果を出力する処理をmainメソッドに書け

3×3行列の積

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} r & s & t \\ u & v & w \\ x & y & z \end{pmatrix} = \begin{pmatrix} ar+bu+cx & as+bv+cy & at+bw+cz \\ dr+eu+fx & ds+ev+fy & dt+ew+fz \\ gr+hu+ix & gs+hv+iy & gt+hw+iz \end{pmatrix}$$

a → a_{00}	r → b_{00}
b → a_{01}	s → b_{01}
c → a_{02}	t → b_{02}
d → a_{10}	u → b_{10}
e → a_{11}	v → b_{11}
…	…

と置き換えてみよう

課題3.2

- 例題3.5で作成したクラスArray3Dに、
  3行3列の行列の転置行列を求めるメソッド
  int[][] transpose(int[][] a)
  を追加せよ

  転置行列（てんちぎょうれつ、transposed matrix）とはm行n列の行列Aに対してAの(i, j)要素と(j, i)要素を入れ替えたn行m列の行列、つまり対角線で成分を折り返した行列のことである。（Wikipediaより）

- このメソッドに、$a=\begin{pmatrix}1&2&3\\4&5&6\\7&8&9\end{pmatrix}$
  を与え結果を出力する処理をmainメソッドに書け