配列２

前回には、配列の基本的な使い方と拡張 for 文について学んだ。本日は配列に付いての追加の説明として、配列のコピー、文字列配列、ガーベジコレクション、多次元配列について学んでいく。

配列のコピー

配列を用意し、その全ての要素を別の配列にコピーすることを考える。まず、以下に間違った例を示していく。

プログラム例１

```
public class Prog07_01
{
        public static void main(String[] args)
        {
                int[] a = {1, 2, 3, 4, 5};
                int[] b = {6, 5, 4, 3, 2, 1, 0};

                System.out.print("a = ");                          // 配列 a の全要素を表示
                for(int i = 0; i < a.length; i++)
                        System.out.print(a[i] + " ");
                System.out.println();

                System.out.print("b = ");                          // 配列 b の全要素を表示
                for(int i = 0; i < b.length; i++)
                        System.out.print(b[i] + " ");
                System.out.println();

                b = a;                  // 配列 a を b にコピー（？）

                a[0] = 10;              // 配列 a[0]の値を書きかえる

                System.out.println("a を b に代入しました。");
                        System.out.print("a = ");                 // 配列 a の全要素を表示
                for(int i = 0; i < a.length; i++)
                        System.out.print(a[i] + " ");
                System.out.println();

                System.out.print("b = ");                          // 配列 b の全要素を表示
                for(int i = 0; i < b.length; i++)
                        System.out.print(b[i] + " ");
                System.out.println();
        }
}
```

実行結果

```
a = 1 2 3 4 5
b = 6 5 4 3 2 1 0
a を b に代入しました
a = 10 2 3 4 5
b = 10 2 3 4 5
```

配列 a の要素数は 5 で、その各値は{1, 2, 3, 4, 5}であり、配列 b の要素数は 7、その各値は{6, 5, 4, 3, 2, 1, 0}である。「b = a;」で配列 b に配列 a の値をコピー（？）し、「a[0] = 10;」で配列 a[0]の値を書き換えている。この処理では、配列 a が {10, 2, 3, 4, 5}、配列 b が{1, 2,

3, 4, 5}となって欲しいのだが、実際の出力結果を確認すると両方の配列とも{10, 2, 3, 4, 5}となってしまっている。この結果は代入後の配列ａとｂは同じものになっていることを示している。つまり、「代入演算子=による配列変数の代入は全要素のコピーではなく、参照先をコピーする」ため、代入後の配列変数ａとｂは同一の配列本体を参照するという結果が得られる。

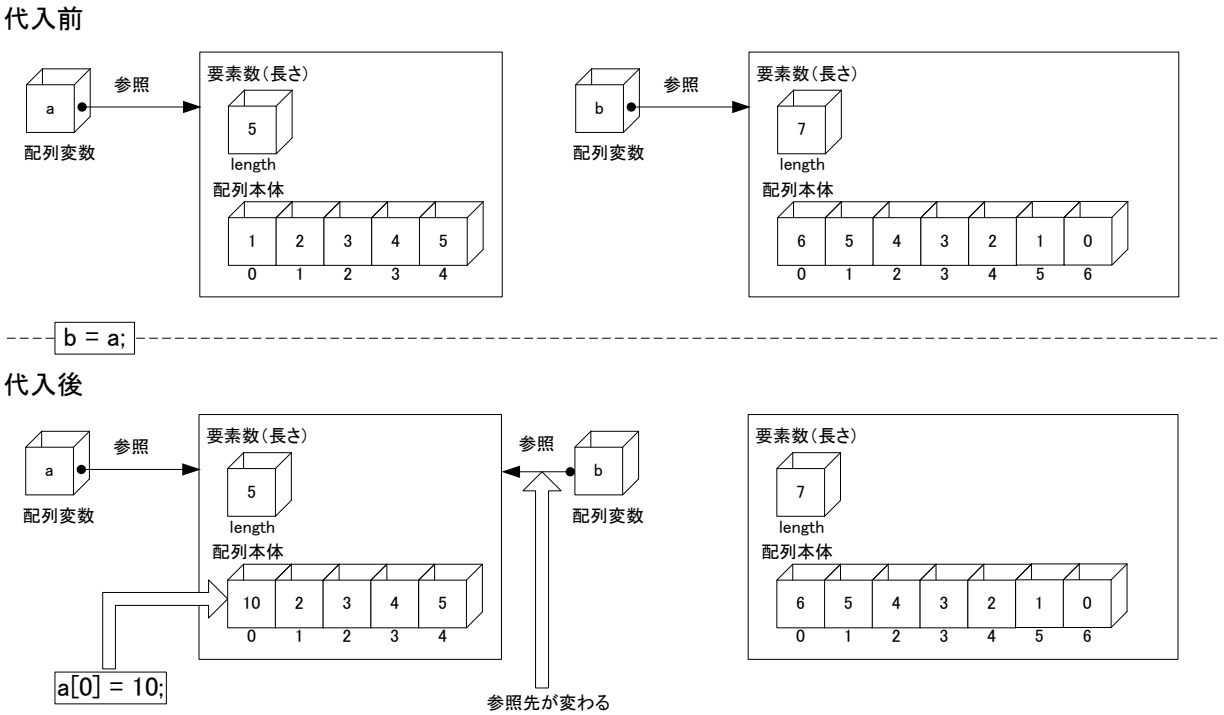

図１　配列変数の代入

そのため、配列をコピーする場合には繰り返し文を使って全要素を一つずつコピーしていく必要がある。この方法を用いて配列をコピーしたプログラム例を以下に示す。

プログラム例２

```
import java.util.Scanner;

public class Prog07_02
{
        public static void main(String[] args)
        {
                Scanner stdIn = new Scanner(System.in);

                System.out.print("要素数:");
                int n = stdIn.nextInt();        // 要素数を読み込む
                int[] a = new int[n];
                int[] b = new int[n];

                for(int i = 0; i < n; i++)      // 配列ａに値を読み込む
                {
                        System.out.print("a[" + i + "] = ");
                        a[i] = stdIn.nextInt();
                }

                for(int i = 0; i < n; i++)      // 配列ａの全要素を配列ｂにコピー
                {
```

```
                        b[i] = a[i];
                }

                System.out.println("a の全要素を b にコピーしました。");

                for(int i = 0; i < n; i++)      // 配列 b を表示
                        System.out.println("b[" + i + "] = " + b[i]);
        }
}
```

実行結果例(斜体は入力値)

```
要素数:5
a[0] = 42
a[1] = 35
a[2] = 85
a[3] = 2
a[4] = -7
a の全要素を b にコピーしました
b[0] = 42
b[1] = 35
b[2] = 85
b[3] = 2
b[4] = -7
```

こちらの例ではキーボードから要素数を n に読み込み、2 つの配列 a、b を生成しているため、どちらの要素数も等しい。もし、要素数が異なる配列でコピーを行うならば、コピー元の配列の要素数を用いて、コピー先の配列の要素数を同じにするため、以下のようにコピー先の配列を生成し直す必要がある。

```
if(a.length != b.length)                // 配列の要素数が異なる場合
        b = new int[a.length];          // 配列を生成し直す

for(int i = 0; i < a.length; i++)       // 配列のコピー
        b[i] = a[i];
```

文字列の配列

文字列は String 型で表すので、その配列の型は String[]型となる。ジャンケンの手を例に取ると、"グー"、"チョキ"、"パー"という文字列の配列が考えられる。要素型は String 型で要素数が 3 の配列を生成すれば良いので以下のようにして、配列を生成し、文字列を代入することで文字列の配列を実現する。

```
String[] hands = new String[3];
hands[0] = "グー";
hands[1] = "チョキ";
hands[2] = "パー";
```

この宣言方法や利用方法はここまでに学んだ int 型や double 型を用いた配列の場合と変わらない。また、以下のように宣言すれば、配列の生成と初期化（代入ではない）も同時に行えることも変わらない。

```
String[] hands = {"グー", "チョキ", "パー"};
```

次のプログラム例は月名の英単語を学習するためのプログラムである。英単語の文字列を配列で用意し、乱数を用いて英単語を選択する。キーボードからの月（数値）の入力を促し、正解したかどうかを判定する。また、間違った場合には正解するまで月の入力を促すことを繰り返す。

プログラム例３

```
import java.util.Random;
import java.util.Scanner;

public class Prog07_03
{
        public static void main(String[] args)
        {
                Random rand = new Random();
                Scanner stdIn = new Scanner(System.in);
                String[] monthString =
                {
                        "January", "Febrary", "March", "April", "May", "June", "July",
                        "August", "September", "October", "November", "December"
                };

                int month = rand.nextInt(12);   // 当てるべき月:0～11の乱数
                System.out.println("問題は" + monthString[month]);

                while(true)
                {
                        System.out.print("何月かな:");
                        int m = stdIn.nextInt();

                        if(m == month + 1) break;
                        System.out.println("違います。");
                }
                System.out.println("正解です。");
        }
}
```

実行結果例(斜体は入力値)

```
問題は August
何月かな: 7
違います。
何月かな: 8
正解です。
```

ガーベジコレクション

以下のプログラム例は配列の値ではなく、配列変数そのものの値を表示するプログラムである。

プログラム例４

```
public class Prog07_04
{
        public static void main(String[] args)
        {
                int[] a = new int[5];                   // ①
                System.out.println("a = " + a);         // ②
```

```
                a = null;                                   // ③
                System.out.println("a = " + a);             // ④
        }
}
```

実行結果例

```
a = [I@ca0b6
a = null
```

①の処理は配列の宣言である。通常の変数はプログラムの実行時に記憶領域が『確保』される。それに対して、new で生成される配列本体は通常の変数と異なり、new が実行されるタイミングで記憶領域が動的に確保される。

配列本体は通常の変数とは異なるため、「**オブジェクト**」と呼ばれる。オブジェクトを指すための変数の型が「**参照型**」である。そのため、配列変数の型である配列型は参照型の一種である。

②では配列変数 a の値そのものを出力した処理だが、その結果は「[I@ca0b6」となっている。配列変数を出力すると特殊な文字の並びが表示される。

③の処理では配列変数 a に null を代入している。null は「**空リテラル**」と呼ばれる。空リテラルが代入された配列変数は「**空参照**」となる。これは配列変数が何も参照していない状態を表す特殊な参照であり、この null の型は「**空型**」と呼ぶ。リテラルとは定数値のことであり、記述する形式によっていくつかの種類があるので、表 1 にまとめておく。

④の処理では null を代入した配列変数 a そのものの値を出力した結果であるが、空参照の配列変数を出力すると「null」と表示されることが確認できる。

表 1　リテラルの種類と型

リテラル名	形式	型	例
整数リテラル	数 先頭に「0」を付けると 8 進数 先頭に「0x」を付けると 16 進数 末尾に「L」か「l」を付けると long 型	int	123 0123 0xFA
		long	123L
浮動小数リテラル	小数点付きの数	double	123.0
	末尾に「F」か「f」を付けると float 型	float	123f
	末尾に「D」か「d」を付けると double 型	double	123d
文字リテラル	シングルクォーテーションで囲む	char	'A'
文字列リテラル	ダブルクォーテーションで囲む	String	"abc"
論理値リテラル	true または false	boolean	true false
空リテラル	null	空型	null

配列変数に null を代入した場合や他の配列本体への参照を代入すると、以下に示す図のように、もともとの配列本体はどこからも参照されない『ゴミ』となってしまう。ゴミをそのままの状態にしておくと、記憶領域の不足をまねく原因となる。そのため、Java ではどこからも参照されなくなったオブジェクト用の領域はその時点で自動的に『解放』され、その領域を再利用できるようにしている。この不要となったオブジェクトの領域を解放して再利用することを「**ガーベジコレクション**」(garbage collection：ゴミ集め) と呼ぶ。

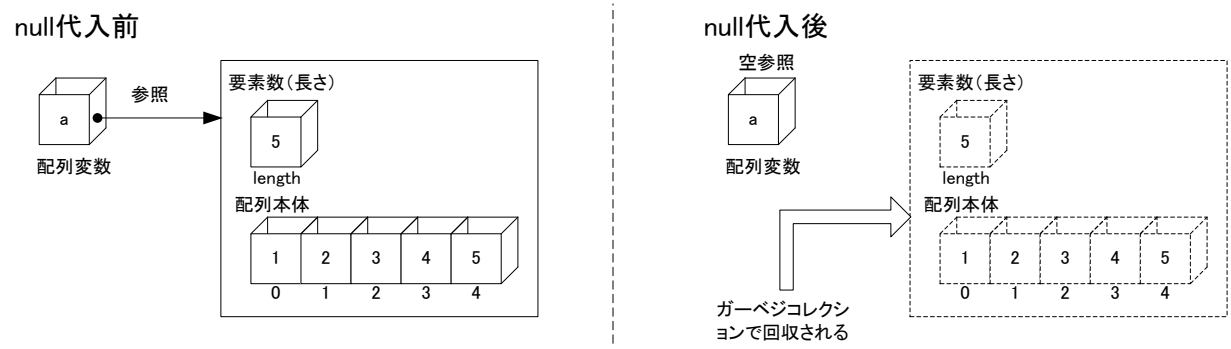

図２　空参照とガーベジコレクション

配列変数に誤って null を代入したり、他の配列本体への参照を代入したりするという状況を防ぐ方法として、以下のように配列変数を final 変数として宣言する方法がある。

```
final int[] a = new int[5];
```

この処理の場合、final 変数として値の書き換えを行うことができなくなるのは配列 a の参照先である。配列の個々の要素の値は書き換えることができる。以下の例を参照のこと。

```
a[0] = 10;              // OK
a = null;               // エラー
a = new int[10];        // エラー
```

多次元配列

ここまでに習った配列は要素が 1 次元で並んでいた。しかし、配列には 2 次元以上のものも存在する。これらは 1 次元の配列と区別するため、多次元配列と呼ばれる。

int 型の 2 次元配列を考えると、その宣言方法は以下の 3 種類が挙げられるが、最も利用されるのは①の宣言方法である。

使用例
①int[][] x; ②int[] x[]; ③int x[][];

例えば、2 行 4 列の配列を宣言と生成を同時に行うには、以下のようになる。

使用例
int[][] x = new int[2][4];

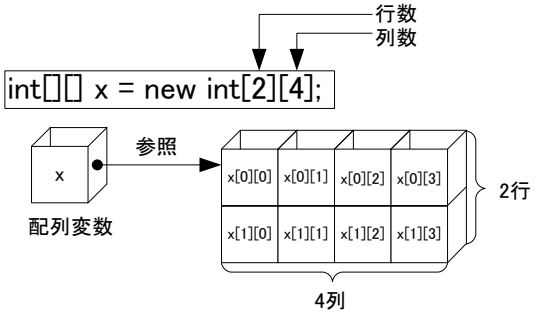

図３　2 次元配列のイメージ

また、以下のようにして、2 行 4 列の 2 次元配列を用意することもできる。

使用例

int[][] x;
x = new int[2][];
x[0] = new int[4];
x[1] = new int[4];

この場合に生成される 2 次元配列の構造は使用例「int[][] x = new int[2][4];」に示した方法で生成した場合の構造と全く同じである。それでは、この 2 次元配列の内部構造を示していく。

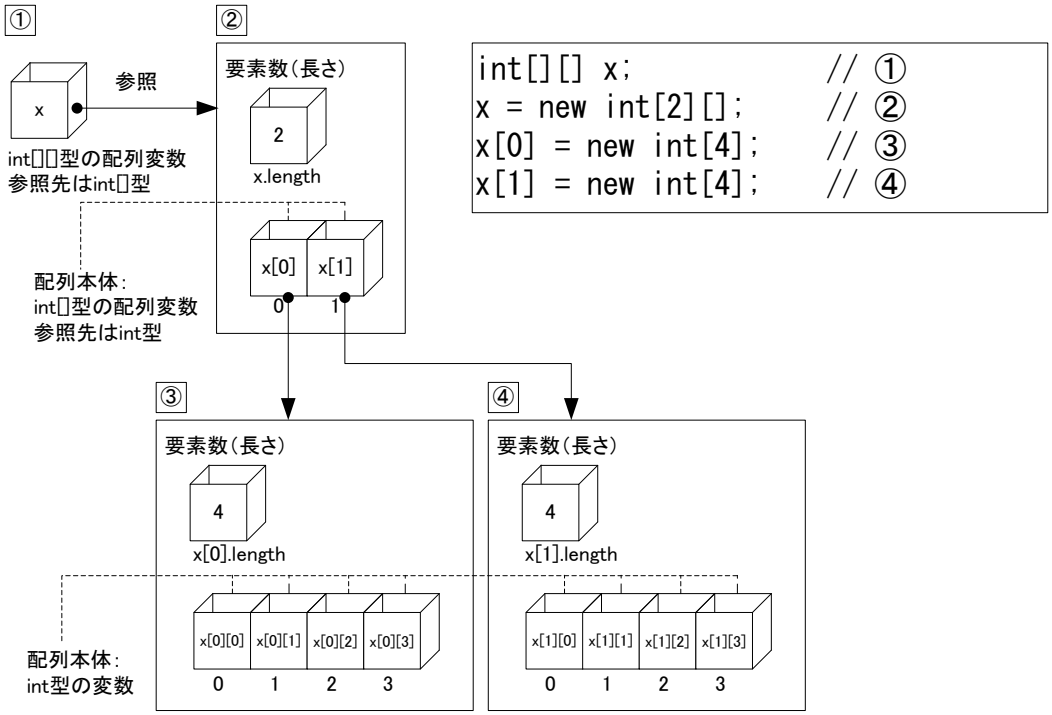

図４　2 次元配列の内部構造

図４において、処理①では int[]型の配列変数を参照する int[][]型の配列変数 x を宣言する。処理②で x の要素数を 2 と定めた。②で生成される配列本体は int 型の変数を参照する int[]型の配列変数 x[0]、x[1]である。実際に int 型の値の出し入れを行うことができる要素は次の③、④の処理で配列本体を生成した後に確保される。ここでは、それぞれ要素数を 4 として生成した。

また、この③、④の処理において、以下に示すように要素数を異なる数で設定すると、凸凹な配列を用意することもできる。

使用例

int[][] c;
c = new int[3][];
c[0] = new int[5];
c[1] = new int[3];
c[2] = new int[4];

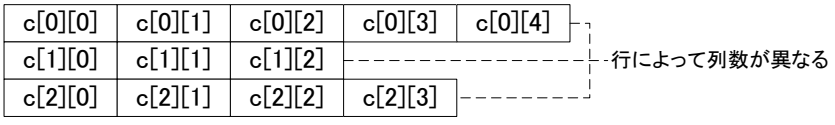

図５　異なる列数を持つ配列のイメージ

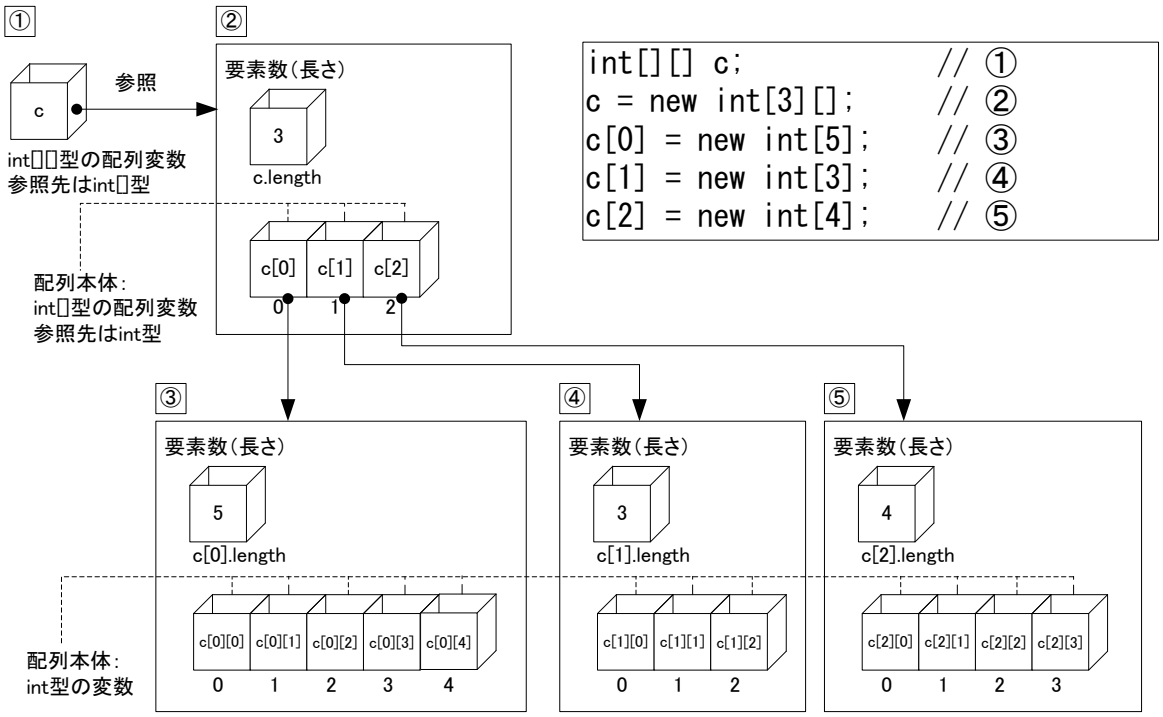

図６　異なる列数を持つ配列の内部構造

ここまでに多次元配列の構造の説明を述べた。複雑な印象を受けたかもしれないが、実際に使用する場合にはシンプルに書くことができる。以下に２次元配列を利用したプログラム例を示す。このプログラムは２次元配列を生成し、乱数で全ての要素に値を代入し、結果を表示している。

プログラム例５

```
import java.util.Random;
import java.util.Scanner;

public class Prog07_05
{
	public static void main(String[] args)
	{
		Random rand = new Random();
		Scanner stdIn = new Scanner(System.in);

		System.out.print("行数:");
		int h = stdIn.nextInt();		// 行数を読み込む

		System.out.print("列数");
		int w = stdIn.nextInt();		// 列数を読み込む

		int[][] x = new int[h][w];

		for(int i = 0; i < h; i++)
			for(int j = 0; j < w; j++)
			{
				x[i][j] = rand.nextInt(100);
				System.out.println("x[" + i + "][" + j + "] = " + x[i][j]);
			}
	}
```

```
}
```

実行結果例(斜体は入力値)

```
行数:2
列数:4
x[0][0] = 72
x[0][1] = 68
x[0][2] = 6
x[0][3] = 6
x[1][0] = 59
x[1][1] = 5
x[1][2] = 18
x[1][3] = 59
```

以下に列数が異なる2次元配列を利用したプログラムを示す。配列の全要素には値を代入していないが、生成時に既定値である0で初期化が行われていることが確認できる。

プログラム例６

```
public class Prog07_06
{
        public static void main(String[] args)
        {
                int[][] c;
                c = new int[3][];
                c[0] = new int[5];
                c[1] = new int[3];
                c[2] = new int[4];

                for(int i = 0; i < c.length; i++)
                {
                        for(int j = 0; j < c[i].length; j++)
                                System.out.printf("%3d", c[i][j]);
                        System.out.println();
                }
        }
}
```

実行結果

```
0 0 0 0 0
0 0 0
0 0 0 0
```

演習

プログラム例１～６を作成し、実行しなさい。

課題４の続き

kadai4_4　配列ａの全要素を配列ｂに逆順にコピーし､結果を出力するプログラムを作成しなさい。二つの配列の要素数は同一であるとみなして良い。

kadai4_5　要素型が int 型である配列を作り、全要素を 1 から 10 の乱数で埋め尽くし、結果を出力するプログラムを作成しなさい。要素数はキーボードから読み込むものとし、連続した要素が同じ値とならないようにすること。たとえば、{1, 3, 5, 5, 1, 2}とならないようにする。配列の要素数は 10 以下とする。同じ値が連続したら、乱数を生成し直すなどで対応する

kadai4_6　2行3列の行列a、bを用意する。代入する値は任意である。行列和を求め、結果を出力するプログラムを作成しなさい。