目次

★5　配列（承前）

★5.6　配列の配列 (p.48)

★5.6.1　配列の配列を作ろう

Java では，多次元配列は「配列の配列」として作ることになる．以下の (3) が示すように，長さの異なる配列をまとめた多次元配列を作ることもできる．

多次元配列の例 (1)

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

for(int i = 0; i < a.length; i++){
    for(int j = 0; j < a[i].length; j++){
        a[i][j] = 100 * (i + 1) + j + 1;
    }
}
for(int i = 0; i < a.length; i++){
    for(int j = 0; j < a[i].length; j++){
        System.out.print(a[i][j] + " ");
    }
    System.out.println("");
}
```

こんなんもあり (3)

```
int[][] a = new int[3][];

for(int i = 0; i < a.length; i++){
    a[i] = new int[i+2];
    for(int j = 0; j < a[i].length; j++){
        a[i][j] = 100 * (i + 1) + j + 1;
    }
}
      (以下上と同じ)
```

こんな作り方もあり (2)

```
int[][] a = new int[3][];

for(int i = 0; i < a.length; i++){

    a[i] = new int[4];

    for(int j = 0; a[i].length; ...
      (以下左と同じ)
```

Q1. 上記の (1),(3) の実行結果を答えなさい．

★5.6.2　配列の配列の初期化

一次元の配列と同様に，多次元配列も初期化することができる．

多次元配列の初期化の例

```
int[][] a = {{2,5,4}, {7,2,1}};  // 2 x 3 の配列
int[][] b = {{2,5,4}, {7}};      // 要素の個数が異なる例
int[][] c = {{2,5,4}, {}, {7}};  // 大きさ 0 の配列もあり
d = new int[2];
int[][] e = {{2,5,4}, d};        // こんなんもあり
```

★6 プリミティブ型と演算子 (p.53)

★6.1 プリミティブ型

プリミティブ型の一覧は p.53 参照．C 言語と異なる点がいくつかあることに注意（byte 型や boolean 型が存在すること，int 型や char 型のビット数，など）．

G06Primitive01.java の一部（println は省略）

```
 4  byte b = 100;
 5  short s = 4096;

 7  int x = 123, y = 0xf, z = 011;


10  long l  = 2147483647  + 1;
11  long l1 = 2147483647L + 1;


14  float f = 0.123f, g = 1.23e-1f;


17  double d = 234.5, e = 2.345e+2;


20  boolean b1 = true;


23  char c = 'a', c1 = '\n';
24  char c2 = 'あ', c3 = '\u266a';
```

実行結果

★6.2 数の演算，変数への代入とキャスト (p.54)

次のようなソースをコンパイルすると，コンパイルエラーとなる．

G06Primitive02.java の一部

```
3   byte b;
4   b = 100;   System.out.println(b);
5   b = b + 1; System.out.println(b);
```

コンパイル結果

```
$ javac G06Primitive02.java
G06Primitive02.java:5: 精度が落ちている可能性
検出値   : int
期待値   : byte
        b = b + 1; System.out.println(b);
                   ^
エラー 1 個
```

その理由を理解するためには，次の二つのことを知る必要がある (☆ 1)．

●演算の際に型が自動的に変換されてしまう場合がある

+,*などの二項演算において二つの数が共に byte,short,char,int 型のいずれかであった場合，int 型で演算を行い，値も int 型となる (☆ 2)．二つの数の一方が long,float,double 型のいずれかであった場合，下記に示すプリミティブ型の大小関係 (☆ 3) に従って二つの数の型の大きい方に合わせて演算を行い値もその型になる (☆ 4)．

```
byte < short < int < long < float < double
```

● 代入文（ 変数 = 式 ;）において，変数の型が式の値の型のよりも「大きい」場合はよいが，変数の型が式の値の型より「小さい」場合は式の値を明示的に型変換（**キャスト**）する必要がある

☆ 1) 後述の G06TypeCast の 6 行目がコンパイルエラーとなるのも同じ理由である．

☆ 2) 上記の例の b + 1 の演算は，b の値を int 型に変換してから int 型で行われる．

☆ 3) 大きい型ほど扱える数の範囲が広い．char 型については p.55 の注釈参照．

☆ 4) 例： double 型+int 型の結果は double 型．

代入文において変数の型の方が大きい場合，式の値の型を変数の型に自動的に変換して代入してくれる．しかし，大小が逆の場合や自動的には変換されないような型変換をしたい場合には，次のような構文を用いて型を明示的に変換する必要がある．

(型名) 式

上記のソースの 5 行目の場合，(1)`b + 1` の演算は `b` の値を型変換して `int` 型で行われ，(2) その結果式の値が `int` 型となるために，キャストなしで `byte` 型の変数に代入できない．エラーとならないようにするためには，`b = (byte)(b + 1);` とすればよい（`b+1` の値を `byte` 型に変換）．

G06TypeCast.java の一部

```
 4  /*** (1) ***/
 5  int   x = 10;
 6  short y = x;  // コンパイルエラー．(short)x とすれば ok
 7  short z = 10; // 10 は int 型だけど定数なので許される
 8  System.out.println("(1) "+x+" "+y+" "+z);
 9
10  /*** (2) ***/
11  double a = 1/2;           // 1/2 の値は int 型の 0
12  double b = 1.0/2;         // double で除算するので 0.5
13  double c = (double)1/2;   // ((double)1)/2 という意味
14  double d = (double)(1/2); // int で除算してから double へ変換
15  System.out.println("(2) "+a+" "+b+" "+c+" "+d);
16
17  /*** (3) ***/
18  int p = Math.random() * 100;  // コンパイルエラー
19  int q = (int)Math.random() * 100;   // コンパイルは通るけど...
20  int r = (int)(Math.random() * 100); // r は ?? 以上 ?? 以下の整数
21  double s = Math.sin(2);  // sin() は引数に double 型の値を受け取る（p.28）.
22  // 2 は int だが定数なので double に自動変換されて sin() に渡される．
23  System.out.println("(3) "+q+" "+r+" "+s);
```

Q2. 下記の (1)–(3) の各行がコンパイルエラーになるかどうか判定し，ならない場合はかめの前進距離を答えなさい．

G06TypeCast2.java の一部

```
 9  double t = Math.sin(Math.PI/6);
10  m.fd(100*t);        // (1)
11  m.fd(100*(int)t);   // (2)
12  m.fd((int)(100*t)); // (3)

15  int d = 100;
16  m1.rt(90); m1.fd(d); m1.lt(90); m1.fd(d); m1.lt(135);
17  m1.fd(???);   // (4)
```

Q3. 上記のプログラムは，かめ `m1` に 3 辺の長さが $1:1:\sqrt{2}$ の直角三角形を描かせようとしている．(4) の”???” の箇所に式を入れて $d \times \sqrt{2}$ 進ませるにはどうしたらよい？

ヒント: p.28. `Math` クラスのクラスメソッド `sqrt` は `double` を返すけれど `Turtle` の `fd` が引数として受け付けるのは `int` だから…

★6.3 式の評価と副作用 (p.57)

これまでみてきたように，**式**が表す演算を行うと結果として値が得られる．演算などを実行して式の値を求めることを，式を**評価する**と言う．例えば，`1/2` という式を評価すると `0` という値が得られ (☆ 5)，`Math.sqrt(100.0)` という式を評価すると `10.0` という値が得られる．

☆5) 1 も 2 も int 型だから，除算は int 型で行われる．`1.0/2` なら値は `0.5`.

式の中には，その式を評価することによって何らかの状態変化が引き起こされるようなものもあり，そのような式は**副作用**を持つという．例えば，`m.moveTo(100, 0)` という式を評価すると，(100,0) までの移動距離という値が得られるとともに「かめが (100,0) に移動する」という副作用が生ずる (☆ 6)．戻り値の型が `void` のメソッド呼び出し式（例えば `m.fd(100)`）は値を持たず副作用のみを持つ．一方，`1/2` や `10+x` といった式は副作用を持たない．

☆6) 「副作用」という言葉から誤解しやすいが，この例からもわかるようにプログラミング言語における副作用は決して「余計な」作用ではない

副作用を起こす式が組合わさった式の場合，部分式の評価の順序が違えば異なる実行結果となる場合がある．例えば，かめ m が (0,0) にいるときに

```
m.moveTo(100, 0) + m.moveTo(100, 100)
```

という式を評価すると「m がまず (100,0) に移動し，次に (100,100) に移動する」という副作用が起こり，この式の値は 200 となる (☆ 7)．これは Java では**基本的に式は左から右へ評価される**ためである．`moveTo()` の順序を入れかえて

☆ 7) p.14 にあるように `moveTo` メソッドは動いた距離を値として返すので，式の値は $100 + 100 = 200$ となる．

```
m.moveTo(100, 100) + m.moveTo(100, 0)
```

とすると，違う実行結果となる．この例では，式の値も上と異なり，241 となる．

実は，代入文によって変数の値が変化するのは，そこで用いられる代入式 (☆ 8) の副作用である．代入式は副作用だけでなく値も持っており，例えば，`x = 0` という代入式は，`x` を `0` にするという副作用とともに，代入された値そのものすなわち `0` という値を表す．そのために，

☆ 8) 代入式： 変数 = 式

```
y = x = 0
```

という式で `x` も `y` も `0` を代入することができる (☆ 9)．`if` 文の条件部に次のような書き方ができるのも同様の理由による (☆ 10)．

☆ 9) よく考えると二つの代入式の評価順序がおかしいと思うかもしれないが…詳細は次節

```
if( (y = m.moveTo(0, 0)) > 100 )
```

☆ 10) この場合，代入式を囲む () は必須．その理由は次節参照．

演算子`&&`と`||`については，式の評価に関して注意が必要である．A `&&` B（A かつ B）という式の評価では，A が `false` ならば B がどちらでも式の値は `false` となるので，この場合には B は評価されない（サボってしまう）．A `||` B（A または B）では，A が `true` の場合には B は評価されない．したがって，次の `if` 文は安全に（0 除算例外を発生させずに）実行できる．

```
if( x != 0 && 10/x == 0)
```

B が評価されない場合には当然その副作用も生じない．勘違いしてバグを生みやすいところである．

Q4. 次の (1),(2) のうちエラーとなるのはどちらか．その理由も答えなさい．

(1)
```
int i = 0; int[] a = {1, 2, 3};
while(i < a.length && a[i] != 0) i++;
```

(2)
```
int i = 0; int[] a = {1, 2, 3};
while(a[i] != 0 && i < a.length) i++;
```