3　反復・配列

計算機が人間に比べて得意なことのひとつに、「同じ作業を正確に反復する」という点があげられるかと思います。本章では、反復処理を含むプログラムについて解説します。

3.1　【サンプルプログラム】戦争ゲーム

早速ですが、まずはサンプルプログラムです。トランプで「戦争」というゲームがあります。2 人が向かい合って、一斉にカードを出し、数字の大きかったほうが勝ち、という勝負を延々と繰り返すゲームです。以下のサンプルプログラムは、その「戦争」に類似したルールで、2 人のカードの枚数を争うゲームです。

このプログラムでは、まず 3,4,5 行目で repeat, awin, bwin という 3 つの変数を宣言します。ここで repeat は繰り返し回数、awin は人物 A が勝った回数、bwin は人物 B が勝った回数を記録するものです。

6 行目の for 文は、変数 repeat に記録された回数だけ反復することを意味しています。

7,8 行目は、トランプで 1 から 13 までのいずれかの値を 1 枚選ぶような数式を意味します。ここで変数 a は人物 A が出したトランプの値、変数 b は人物 B が出したトランプの値で、いずれも 1 から 13 の間の整数となるようになっています。

ここで、この算出式について説明します。まず

$$\text{Math.random()}$$

は 2.1 節でも示したとおり、0.0 から 1.0 の間で乱数を生成します。続いて

$$\text{(int)(13.0 * Math.random())}$$

では、この 0.0 から 1.0 の間の乱数に 13.0 を掛けて、それを小数点以下切り捨てて整数にします。そのため上記の式の値は、0 から 12 の間の整数になります。そして

$$\text{int a = (int)(13.0 * Math.random()) + 1;}$$

では、これに 1 を加えた値を変数 a 代入しています。よって、変数 a の値は 1 から 13 の間の整数、つまりトランプの数字と同じ範囲の整数ということになります。そして変数 b の値も同様に、1 から 13 の間の整数になります。

続いて、9 行目以降の if 文および else 文では、

- $a > b$ であれば人物 A の勝ちであることを示し、変数 awin に 1 を加える。
- $a < b$ であれば人物 B の勝ちであることを示し、変数 bwin に 1 を加える。
- いずれでもなければ、引き分けであることを示す

のいずれかの処理が実行されます。

最後に 22 行目で、ゲームが終了し、人物 A および人物 B が何勝したか、を表示します。

```
public class Senso {
   public static void main(String[] args) {
     int repeat = 20;
     int awin = 0;
     int bwin = 0;
     for(int i = 0; i < repeat; i++) {
       int a = (int)(13.0 * Math.random()) + 1;
       int b = (int)(13.0 * Math.random()) + 1;
```

```
        if(a > b) {
          System.out.println("A wins !!  a=" + a + " b=" + b);
          awin++;
        }
        else if(a < b) {
          System.out.println("B wins !!  a=" + a + " b=" + b);
          bwin++;
        }
        else {
          System.out.println("draw a=b=" + a);
        }
      }
      System.out.println("GAME OVER ...  A wins " + awin + " times, B wins " + bwin + " times");
    }
}
```

では、以上のプログラムを Senso.java というファイルに保存し、実行してみてください。あくまでも一例として、以下のような表示が現れるはずです。

```
B wins !!  a=8 b=11
B wins !!  a=7 b=9
A wins !!  a=7 b=1
draw a=b=8
A wins !!  a=13 b=11
A wins !!  a=5 b=1
B wins !!  a=7 b=13
B wins !!  a=2 b=7
B wins !!  a=3 b=7
A wins !!  a=12 b=7
A wins !!  a=13 b=12
A wins !!  a=9 b=3
B wins !!  a=11 b=13
B wins !!  a=5 b=7
A wins !!  a=8 b=6
A wins !!  a=10 b=3
B wins !!  a=7 b=9
A wins !!  a=5 b=2
A wins !!  a=12 b=1
B wins !!  a=5 b=13
GAME OVER ...  A wins 10 times, B wins 9 times
```

3.2 for 文による反復

反復処理の中でも、すでに反復回数が定義されているものについては、for という文法を用いて記述します。一例として、以下のような計算を考えてみましょう。

$$sum = \sum_{i=1}^{10} i$$

この式は、1 から 10 までの整数の総和を求める式です。この計算は、以下の式を 1 から 10 まで反復するのと等価であると考えられます。

$$a_i = a_{i-1} + i \text{（ただし } a_0\text{=0，} a_i \text{ は 1 から } i \text{ までの整数の総和）}$$

それでは、上記の式を実現するプログラムを以下に紹介します。

```
public class For {
   public static void main(String[] args) {
     int a = 0;
     for(int i = 1; i <= 10; i = i + 1) {
       a = a + i;
     }
     System.out.println(a);
   }
}
```

このプログラムでは、まず 3 行目で変数 a を 0 に初期化します。続いて 4 行目の for 以下で、反復を指定します。for の次のカッコの中は、;(セミコロン) で 3 つの文に区切られています。この 3 つの文は、以下の意味を持ちます。

- 最初の文は、**反復の初期状態** を記述します。以下のプログラムの場合、int i=1 ということで、整数型の変数 i を用意し、その初期値を 1 とします。
- 2 つめの文は、**反復条件** を記述します。ここに記述された条件が成り立つ限り、処理は反復されます。以下のプログラムの場合、i<=10 ということで、i が 10 以下である限り反復されます。ここで使用できる条件記述の算術記号については、表 3 を参照してください。
- 3 つめの文は、**反復ごとに毎回行う処理** を記述します。以下のプログラムの場合、i = i + 1 ということで、i に 1 を加算した値を、今までの i の値に上書きするように代入する、という処理を行います。

反復の対象となる範囲は、for の後にある { と } で括られた範囲です。つまり、以下のプログラムでは、a = a + i という処理が反復されます。
これらをまとめると、

for(○○; ▽▽; △△) { □□ }

という文は、

1. 最初に○○という処理を行う。
2. ▽▽という条件が成立しなければ反復終了。成立すれば 3. へ。
3. □□という処理を実行する。
4. △△という処理を行って 2. に戻る。

という反復処理をすることに相当します。

続いて以下のプログラムの 5 行目にある `a = a + i` は、a に i を加算した値を、今までの a の値に上書きするように代入する、という処理を行います。

それでは、このプログラムを `For.java` というファイル名で保存して、実行してみましょう。以下のような表示結果が表示されれば、正常に動作しているといえましょう。

```
55
```

同一の変数への代入文

上述のプログラムの `a = a + i` という代入文のように、同一の変数（この場合）に今までの値を上書きするような代入文の場合、`a += i` というように記述することもできます（むしろ、このような記述を推奨されています。）。

Java 言語での四則演算における同様な記述をまとめて、**算術代入演算子**といいます。算術代入演算子の代表的なものを、表 5 に列挙します。

表 5: Java 言語における算術代入演算子。

記号の種類	記号の説明
`a += b`	a に b を加算する
`a -= b`	a に b を減算する
`a *= b`	a に b を乗算する
`a /= b`	a に b を除算する

3.3 while 文による反復

反復処理の中でも、反復回数が定義されていないものについては、`while` という文法を用いて記述します。一例として、

1 から i（i は $i > 1$ である整数）までの総和 a を求める処理を、
$a \leq 100$ である限り反復する

という処理を実現するプログラムを紹介します。

```
public class While {
   public static void main(String[] args) {
     int a = 0;
     int i = 1;
     while(a <= 100) {
       a += i;
       i += 1;
     }
     System.out.println("a=" + a + " i=" + i);
   }
```

}

このプログラムでは、まず3行目で変数 a の値を 0 に、4 行目で変数 i の値を 1 に初期化します。続いて 5 行目の `while(a <= 100)` ですが、これは数式 $a \leq 100$ が成立する限り { と } の間を反復する、という意味を持ちます。つまり、$a \leq 100$ である限り、6 行目の `a += i` および 7 行目の `i += 1` を反復します。言い換えれば、

`while(` ○○ `) {` □□ `}`

という構文は、

○○という条件が成立する限り、□□という処理を反復する

という意味だと思ってください。

なお、`while` 文の `()` の中で使用できる条件記述の算術記号については、表 3 を参照してください。

続いて 9 行目の `System.out.println("a=" + a + " i=" + i)` ですが、今までにも出てきているように、`System.out.println` という命令は画面出力のために用いられます。この後のカッコの中に注目して下さい。この意味ですが、

- `"a="` という文字列
- 変数 a の値
- `" i="` という文字列
- 変数 i の値

を左から順に表示しなさい、という意味を持ちます。この 4 つの間にある+は、数式の加算という意味ではなく、これらの情報を左から並べて記述しなさい、という意味を持ちます。

それでは、このプログラムを `While.java` というファイル名で保存して、実行してみましょう。以下のような表示結果が表示されれば、正常に動作しているといえましょう。

```
a=105 i=15
```

1 を加える、または 1 を減じる

上述のプログラムの `i = i + 1` という代入文のように、同一の変数（この場合）に今までの値に 1 を加算して上書きするような代入文の場合、`a++` というように記述することもできます（むしろ、このような記述を推奨されています。）。同様に、1 を減算して上書きするような代入文の場合、`a--` というように記述することが推奨されています。 これらをまとめて**単項演算子**と呼びます。単項演算子の代表的なものを、表 6 に列挙します。

表 6: Java 言語における単項演算子。

記号の種類	記号の説明
`a++`	a に 1 を加算する
`a--`	a に 1 を減算する

3.4 do-while 文による反復

while 文に非常に似た構文で、do-while 文というものがあります。これは

do { □□ } while(○○)

という構文で、while 文と同様に、

○○という条件が成立する限り、□□という処理を反復する

という意味を有します。ただし、while 文が { と } の間を実行する前に () の中の条件を判断するのに対して、do-while 文では { と } の間を実行した後に () の中の条件を判断します。つまり do-while 文では、反復の最初の 1 回は、{ と } の間を無条件に実行します。

さきほどの while 文のプログラムを do-while 文に書き換えると、以下のようになります。

```
public class DoWhile {
   public static void main(String[] args) {
     int a = 0;
     int i = 1;
     do {
       a += i;
       i += 1;
     } while (a <= 100);
     System.out.println("a=" + a + " i=" + i);
   }
}
```

3.5 配列

この章の冒頭で、数列を例にとって for 文を紹介してきました。Java 言語では、数列のように、通し番号をもって一連の情報を格納するために、配列という仕組みを用意しています。

配列では、変数名のあとに [] という記号を設け、その [と] の中に通し番号を記入します。例えば

$$a_i = 3, 2, 5, 7 \qquad (i = 1, 2, 3, 4)$$

という数列があったとします。このとき、この数列の個々の要素は、

$$a_1 = 3, a_2 = 2, a_3 = 5, a_4 = 7$$

というように記載されます。Java 言語では、これと同様の内容を

a[0]=3; a[1]=2; a[2]=5; a[3]=7;

というように記載します。ここで注意すべき点は、数列の添え字は 1 から始まるのが一般的なのに対して、**Java 言語の配列の通し番号（インデックス）は 0 から始まるように規定されている**という点です。この点は非常に間違えやすいので、くれぐれも注意して下さい。

では、この章の冒頭にあった For クラスのプログラムを少し書き換えて、配列を使ったプログラムの例を紹介します。

このプログラムでは、まず 3 行目で、変数 aa を配列として宣言します。配列を宣言する際には、以下の点を守ってください。

- 左辺にて宣言する変数（この場合 aa）の後の [] の中は、宣言時には空欄とします。
- 右辺では new + 変数の型 という形の構文を用いて、[] の中には必要な個数（この場合 11 個）を記載します。

つまり、このプログラムでは、aa[0] から aa[10] までの、合計 11 個の変数を配列として用いることになります。

続いて 4 行目で、aa[0] に 0 を代入します。この時点では、aa[1] から aa[10] には何の値も代入されていないことに注意して下さい。

続いて 5 行目以降の for 文で、

$$a_i = a_{i-1} + i \text{（ただし } a_0\text{=0，} i \text{ は 1 から 10 までの整数）}$$

という処理を、i=1 から i=10 まで反復します。これによって a[i] には上記の a_i の値が代入され、続いて System.out.println(aa[i]) によってその値が出力されます。

```
public class For {
   public static void main(String[] args) {
     int aa[] = new int[11];
     aa[0] = 0;
     for(int i = 1; i <= 10; i++) {
       aa[i] = aa[i - 1] + i;
       System.out.println(aa[i]);
     }
   }
}
```

では早速、このプログラムを実行してみて下さい。以下のように画面表示されれば、正しく実行されています。

```
1
3
6
10
15
21
28
36
45
55
```

3.6 continue 文と break 文

いままで習った反復処理は、for 文も、while 文も、do-while 文も、全て反復続行の判定を所定の位置にて実行します。それに対して、反復処理中の任意の場所で反復処理を省略したり終了したりするために、continue 文や break 文を使います。

反復処理を囲む { と } の間で、continue 文 が実行されたときには、プログラムは反復処理中の continue 文より後の処理を省略し、反復処理の冒頭に戻ります。

例えば For クラスを以下のように書き換えたとします。このとき、7 行目の if 文で i % 2 == 1 が成立するとき、つまり i が奇数のとき、continue 文が実行されます。このとき、ここより後にある System.out.println(aa[i]); は省略され、5 行目の for 文に戻ります。

さて、実行結果はどのようになるか、想像できますでしょうか？

```
public class For {
   public static void main(String[] args) {
     int aa[] = new int[11];
     aa[0] = 0;
     for(int i = 1; i <= 10; i++) {
       aa[i] = aa[i - 1] + i;
       if(i % 2 == 1) {
         continue;
       }
       System.out.println(aa[i]);
     }
   }
}
```

反復処理を囲む { と } の間で、break 文 が実行されたときには、プログラムは反復処理を終了し、反復処理を囲む } の後にジャンプします。

例えば For クラスを以下のように書き換えたとします。このとき、7 行目の if 文で aa[i] > 40 が成立するとき、break 文が実行されます。このとき、ここから } の後にジャンプし、このままプログラムは実行終了します。

さて、実行結果はどのようになるか、想像できますでしょうか？

```
public class For {
   public static void main(String[] args) {
     int aa[] = new int[11];
     aa[0] = 0;
     for(int i = 1; i <= 10; i++) {
       aa[i] = aa[i - 1] + i;
       if(aa[i] > 40) {
         break;
       }
       System.out.println(aa[i]);
     }
   }
}
```