4　クラスとメソッド

いままでのサンプルプログラムでは、いずれも最初の 2 行が、以下のような書き出しになっていましたが、この意味について詳しく説明していませんでした。

```
public class Bmi {
   public static void main(String[] args) {
```

本章では、この 1 行目にある「class」の意味と、2 行目にある「main」の意味を中心に、解説を進めます。

4.1　【サンプルプログラム】当たり付きアイス

本章も例によって、サンプルプログラムから始めましょう。ここでは「当たりが出たらもう 1 本食べられるアイス」というものを考えてみましょう。皆さんはこのような当たりつきのアイスを食べるとき、食べ終わりそうになったら、アイスのバーを見るはずです。そしてバーに「当たり」と書いてあったら、もう 1 本アイスをもらうでしょう。そして 2 本目のアイスも同様に、食べ終わりそうになったら、バーを見るはずです。そしてそのバーも「当たり」と書いてあったら…以後、強運がいつまでも続いていたら、皆さんは何本でもアイスを食べられることになるわけです。

このような「当たりつきのアイス」で何本連続「当たり」を出せるか、というプログラムの例をお見せします。

```
   public class Recursive {
     public static void eatIce() {
       System.out.println("Eat one ice ...");
       double value = Math.random();
       if(value >= 0.75) {
         System.out.println(" ...  You are lucky !!  One more ice !!");
         eatIce();
       }
     }

     public static void main(String args[]) {
       eatIce();
       System.out.println(" ...  See you.");
     }
   }
```

では、このプログラムを Recursive.java というファイル名で保存して、実行してみてください。運がよければ、以下のように、たくさんアイスを食べられるはずです。

```
Eat one ice ...
...  You are lucky !!  One more ice !!
Eat one ice ...
```

```
...  You are lucky !!  One more ice !!
Eat one ice ...
...  See you.
```

本章では「クラス」「メソッド」という単語を紹介します。これらの単語は Java 言語において抽象的な概念を意味するもので、慣れるまでは理解し難いかもしれません。しかし、これらの概念は大きなプログラムを開発する上で非常に重要ですので、頑張って理解して下さい。

このプログラムでは 2 行目に、eatIce という名前のメソッドがあります。これを呼び出すと、まず 3 行目で Eat one ice ... と表示されます。これは文字通り、「アイスを 1 個食べます」という意味になります。続いて 4 行目で、変数 value に 0 から 1 の間の実数が代入されます。この値が 0.75 以上であればプログラムは、アイスが当たりだとみなします。このとき... You are lucky !! One more ice !!と表示され、もう一度 eatIce が呼び出され、2 行目に戻ります。この値が 0.75 未満であれば、もう一度 eatIce が呼ばれることはなくなり、現在 eatIce を呼び出した元に戻ります。

この eatIce メソッドは、11 行目の main メソッドから呼び出されています。そして main メソッドに戻ると、... See you. と表示され、これ以上アイスが食べられないことを意味します。

このプログラムでも採用されている「クラス」「メソッド」について、次節以降で詳しく紹介します。

4.2 クラス

クラスとは Java 言語において、1 個のプログラムを表す単位です。

個々のクラスには、固有の名前をつける必要があります。上のサンプルプログラムのうち

```
public class Bmi {
```

の行は、Bmi という名前をつけられたクラスが定義されている、ということを意味しています。

すでに前述の通り、Java 言語ではクラス名とファイル名が対応していないといけません。その対応は

ファイル名＝<クラス名>.java

です。クラス名およびファイル名は、大文字と小文字を厳密に区別しますので、それも間違えないようにしてください。また.(ピリオド) のあとの「java」は、全て小文字にして、大文字を使わないようにしてください。

なお、文字列を扱うために何度か出現した「String」も、クラスの一種です。

4.3 メソッドと return 文

メソッドとは、プログラム中の一かたまりの処理をまとめた単位です。[5]

プログラムを書くときには、あまり長大な処理を連続的に記述するより、いくつかのブロックに分けて記述するほうが、読みやすいとされています。つまりメソッドは、プログラムを小分けして読みやすくするために用いるのが一般的です。

また、プログラム中の複数の場所で同じ処理をさせたいときには、それをメソッドという単位にまとめておくと、複数の場所で同じ処理を重複して記述することなく、かえってシンプルにプログラムを書くことができます。

[5] C 言語では「関数」という概念を習いますが、これとメソッドは非常に似た、ほぼ同じ概念のものだと思って下さい。

メソッドには **引数** と**返り値** という概念があります。この概念を理解するために、数学に出てくる関数について考えてみましょう。一例として、

$$y = f(x) \tag{1}$$

という関数を考えてみて下さい。この関数では、x という変数値を $f(x)$ という関数に代入すると、y という値が与えられます。これに「メソッド」「引数」「返り値」という単語を当てはめてみると、以下のような関係になります。

表 7: Java 言語における「メソッド」「引数」「返り値」の関係。

記号	数学上の意味	Java 言語での用語
x	代入する値	引数
$f(x)$	計算式	メソッド
y	計算結果	返り値

メソッドを使う単純な例

では、メソッドを使う単純な例として、`Bmi.java` を `BmiMethod.java` というファイル名でコピーして、以下のように書き換えて保存して下さい。

```
public class BmiMethod {
   public static double calculateBmi(double weight, double height) {
     double result = weight / (height * height);
     return result;
   }

   public static void main(String[] args) {
     double w = 67.0;
     double h = 1.78;
     double bmi = calculateBmi(w, h);
     System.out.println("BMI=" + bmi);
   }
}
```

このプログラムでは、2 行目の `calculateBmi` および、7 行目の `main` がメソッドです。メソッドでは、その名前の後ろに、引数を記述する `()` が加わります。

2 行目の `calculateBmi` の引数は、`(double weight, double height)` と書いてありますが、これは `double` 型の `weight` という変数と、`double` 型の `height` という変数の、合計 2 個の変数を有することを意味します。

7 行目の `main` の引数は、`(String[] args)` と書いてありますが、これは `String` クラスの配列の `args` という変数を有することを意味します。

続いて、各々のメソッド名の前に、2 行目には `double`, 7 行目には `void` という単語があることに注目して下さい。2 行目の `double` は、`calculateBmi` メソッドの返り値が `double` 型であることを示していま

す。7 行目の void は、main メソッドの返り値が何もないことを示しています。**void とは「型を持たない」**という意味だと思ってください。

続いて、calculateBmi メソッドの中を見て下さい。

3 行目の double result = weight / (height * height); では、double 型の変数 result に、BMI の計算結果を代入しています。続いて 4 行目の return result; では、変数 result に代入された値をメソッドの外に返します。つまり、

calculateBmi メソッドの返り値は、
return の後に記述される値（つまり、変数 result に代入された値）

ということになります。

さて、calculateBmi メソッドは main メソッドの中で、10 行目の double bmi = calculateBmi(w, h); で呼び出されています。この意味ですが、

メソッド calculateBmi に変数 w と変数 h を引数として渡し、
その返り値を double 型の変数 bmi に代入する

と解釈して下さい。つまり、このプログラムの場合には、

1. main メソッドの変数 w および h の値が、calculateBmi メソッドの変数 weight および height の値として渡される。
2. calcluateBmi メソッドは、変数 weight および height を用いて返り値を求める。
3. calculateBmi メソッドの返り値が、main メソッドの変数 bmi に代入される。

という手順で処理が進んでいる、ということになります。

return 文

では、先ほどのプログラムに出てきた **return** について説明します。return 文には 2 つの意義があります。

まず 1 つ目の意義は、void 型以外のメソッドにおいて、返り値を指定することにあります。void 型以外のメソッドでは、

return 値 ;

という用法によって、そのメソッドの呼び出し元への返り値を指定できます。

続いて 2 つ目の意義は、メソッドの処理の途中であっても、強制的にそのメソッドから抜け出し、呼び出し元に戻ることができることにあります。例えば method1 というメソッドの途中で、強制的に method1 から抜け出したいときには、

```
public static double method1() {
  ...;
  return 0.1;
  ...;
}
```

というように記述すると、return 文のあった位置から method1 を抜け出します。結果として、method1 メソッド中の、return 文より後の処理は、実行されません。

なお void 型のメソッドにおいても、return 文を使ってメソッドの途中で抜け出すこともできます。ただし void 型の場合、返す値はありませんので、

```
public static void method1() {
  ...;
  return;
  ...;
}
```

というように、return の後に何も値を書かずにセミコロン（;）を打つことになります。

再帰

ところで、メソッドの面白い使い方に、**再帰** という考え方があります。再帰とは一言でいうと、Java 言語の場合には、**メソッドが自分のメソッドを呼び出すこと** を指します。本章の冒頭に載せた「当たり付きアイス」のプログラムは、非常に単純な再帰の例といえます。

4.4 複数のクラスを用いるプログラム

複数のクラスを用いるプログラムの作成

Java 言語では、複数のクラスを組み合わせて 1 つのプログラムを構成する、ということが自在にできるようになっています。以下、その簡単な例を紹介します。

まず、以下のコマンドによって、BmiMethod.java を BmiMethod1.java および BmiMethod2.java にコピーして下さい。

```
cp BmiMethod.java BmiMethod1.java
cp BmiMethod.java BmiMethod2.java
```

続いて、BmiMethod1.java を以下のように編集して下さい。

```
public class BmiMethod1 {
  public static void main(String[] args) {
    double w = 67.0;
    double h = 1.78;
    BmiMethod2 method2 = new BmiMethod2();
    double bmi = method2.calculateBmi(w, h);
    System.out.println("BMI=" + bmi);
  }
}
```

さらに、BmiMethod2.java を以下のように編集して下さい。

```
public class BmiMethod2 {
```

```
    public double calculateBmi(double weight, double height) {
      double result = weight / (height * height);
      return result;
    }
}
```

では、以上のプログラムについて説明します。まず BmiMethod1 クラスを見て下さい。このプログラムでは、まず 3,4 行目にて、今までと同様に、double 型の変数 w および h を宣言します。

続いて 5 行目の `BmiMethod2 method2 = new BmiMethod2();` ですが、これは

BmiMethod2 クラスを 1 個確保し、それを変数 method2 に代入する

という意味を持ちます。ここで new という命令は、その後に記述されるクラスの実体（**インスタンス**）を確保する、という意味を持ちます。3.5 節でも、指定された要素数の配列を確保するために new 命令を用いていますが、これと意味は同一であると考えてよろしいかと思います。なお new 命令については、4.5 節でも後述します。

続いて 6 行目の `double bmi = method2.calculateBmi(w, h);` ですが、これは

変数 method2 の中にある calculateBmi メソッドに、
変数 w および h を引数として渡し、
その返り値を変数 bmi に代入する

という意味を持ちます。このように、ある変数のメソッドを呼び出すときには、**変数名. メソッド名 ()** というように、変数名とメソッド名をピリオド (.) で連結して記述します。[6]

続いて BmiMethod2 クラスを見て下さい。このクラスは今までと違って main メソッドがありません。Java 言語のクラスは、main メソッドがなくても成立します。ただしこの場合、Java 言語では main メソッドから実行を開始しますので、main メソッドのないクラスはそれだけでは実行できません。つまり言い換えれば、**main メソッドのないクラスは、main メソッドのあるクラスと組み合わせることで、初めて利用できる**ということを意味します。BmiMethod2 クラスの中にある calculateBmi メソッドについての説明は、4.3 節と同様なので割愛いたします。

複数のクラスを用いるプログラムの実行

では、このプログラムを実行してみましょう。まず以下の通り、2 つのファイルを両方コンパイルして下さい。

```
javac BmiMethod1.java
javac BmiMethod2.java
```

すでに説明したとおり、このプログラムは BmiMethod1 クラスが BmiMethod2 クラスを呼び出しています。このような場合には、プログラムを動かすためには、BmiMethod1 クラスと BmiMethod2 クラスの両方をコンパイルする必要があります。

[6] これとは別に、あるクラスのメソッドを呼び出すときに、**クラス名. メソッド名 ()** という記述方法もあります。これらの詳細については後述します。

では、このプログラムを実行してみましょう。java 命令では、main メソッドを含むクラスを指定しますので、この場合には以下のように、BmiMethod1 クラスを指定します。

```
java BmiMethod1
```

どうでしょうか？ 以下のように表示されましたでしょうか？

```
BMI=21.146319909102385
```

4.5 new 命令とコンストラクタ

続いて、3.5 節および 4.4 節で登場した new 命令について、もう一度解説します。

Java 言語では、プログラムの単位となる「クラス」と、それを実行する際に生成させる「インスタンス」という概念があります。この「インスタンス」は、プログラム中で複数発生させることができます。そのため、どこで何個のインスタンスを発生させるか、という指定をする必要があります。そこでインスタンスを発生させるための命令として、new という命令が用意されています。

new 命令の使い方

new 命令は、以下のような形で使われます。

- 特定のクラスのインスタンスを確保するとき。4.4 節参照。以下のように、右辺に「new クラス名 ()」と記述し、その返り値を左辺の変数に代入する、という使い方が一般的である。

  ```
  BmiMethod2 method2 = new BmiMethod2();
  ```

- 配列を確保するとき。3.5 節参照。この場合はクラスに限らず、int や double などの変数の型も対象となる。以下のように、右辺に「new 変数型 [個数]」と記述し、その返り値を左辺の変数に代入する、という形で多く用いられる。

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

- クラスの配列を確保するときは、要注意。配列そのものを確保するだけでなく、配列の個々に対してクラスを確保する、という処理も必要になる。

  ```
  BmiMethod2 method2[] = new BmiMethod2[10];
  for(int i = 0; i < 10; i++) {
     method2[i] = new BmiMethod2();
  }
  ```

コンストラクタ

特定のクラスのインスタンスを生成する瞬間に、何らかの処理を実行したい場合があります。そこで Java 言語では、new 命令を呼び出したときに実行されるメソッドを記述することができます。この、new 命令を呼び出したときに実行されるメソッドを**コンストラクタ** と呼びます。

コンストラクタは、クラス名にカッコをつける形で記述します。例えばBmiMethod2クラスであれば、コンストラクタはBmiMethod2()と記述します。

一例として、4.4節で紹介したBmiMethod2クラスにコンストラクタを追加してみましょう。

```
public class BmiMethod2 {
   double weight;
   double height;
   public BmiMethod2() {
     weight = 67.0;
     height = 1.78;
   }
   public BmiMethod2(double w, double h) {
     weight = w;
     height = h;
   }
   public double calculateBmi() {
     double result = weight / (height * height);
     return result;
   }
}
```

このプログラムでは、

- 4行目のpublic BmiMethod2()
- 8行目の public BmiMethod2(double w, double h)

の2つのコンストラクタが混在しています。

クラスでは、コンストラクタを2つ以上持つことも可能ですが、1つだけでも構いませんし、全く書かなくても構いません。全く書かない場合には、new命令が呼び出された際には単にインスタンスを生成するだけで、それ以外の処理は何もしません。

ここで4行目のコンストラクタが呼ばれたときには、new命令と同時に、変数weightおよびheightに、それぞれ67.0, 1.78が代入されます。

8行目のコンストラクタが呼ばれたときには、new命令と同時に、引数wおよびhの値が、変数weightおよびheightに代入されます。

では今度は、以下のようにBmiMethod1.java を書き換えてみてください。

```
public class BmiMethod1 {
   public static void main(String[] args) {
     double w = 75.0;
     double h = 1.68;
     BmiMethod2 method2 = new BmiMethod2(w, h);
     double bmi = method2.calculateBmi();
     System.out.println("BMI=" + bmi);
   }
}
```

このプログラムでは、5 行目の BmiMethod2 method2 = new BmiMethod2(w, h); が呼ばれたときに、BmiMethod2 クラスの 8 行目のコンストラクタが呼び出されます。つまり、BmiMethod1 クラスの 3,4 行目に宣言されている変数 w および h の値が使われます。これを実行すると、

```
BMI=26.573129251700685
```

と表示されるはずですが、いかがでしょうか。
今度は上記の BmiMethod1 クラスのうち、5 行目の

```
BmiMethod2 method2 = new BmiMethod2(w, h);
```

を

```
BmiMethod2 method2 = new BmiMethod2();
```

に変えてみてください。これが呼ばれたときには、BmiMethod2 クラスの 4 行目のコンストラクタが呼び出されます。つまり、BmiMethod1 クラスの 3,4 行目に宣言されている変数 w および h の値は使われず、BmiMethod2 クラスの 4 行目のコンストラクタで設定される値が用いられます。これを実行すると、

```
BMI=21.146319909102385
```

と表示されるはずですが、いかがでしょうか。