# Java プログラミング入門

伊藤貴之
お茶の水女子大学

# まえがき

私がプログラミングに初めて接したのは1979年、当時私は小学6年生でした。まだ携帯電話が普及していなかった時代に、当時高校生の親戚が友人との会話にトランシーバーという機械を使用していました。親戚はその専門月刊誌を買っていたのですが、どうも専門的すぎて読む気がしなかったようで、たまたま訪れた私の家に置き忘れてしまいました。そしてその雑誌を偶然にも私が読み、その雑誌の連載記事にBASICというプログラミングの解説が載っていたのに興味をもって、独学を始めたのでした。

小学生の私が、なぜその連載記事からプログラミングを習得できたのか、薄れる記憶をたどって思い出しました。おそらくその一番の秘訣は、それが教科書ではなく商用月刊誌の連載だった、という点にあるような気がしています。月刊誌の中の8ページ程度の短い記事の中で、簡潔に要点を説明してあったこと、そしてサンプルプログラムが身近なものであったことが、私にとって非常に大きな魅力だったように思います。

私はどうしてもそのプログラムを動かしてみたくなり、秋葉原の電気街に一人で行ってみました。そこには店頭展示品のパソコンが数台ありました。私はノートに書きとどめたプログラムを打ち込んで実行し、想定通りに答えが出るのも確認しました。電気街のおじさんは優しくて、私が長い時間にわたって店頭展示品を独占していたのを黙認してくれていました。

私はその月刊誌を通して、プログラミングをはじめとする情報技術に興味を持ち、理工学部に進学してコンピュータ会社に就職し、13年間の企業生活を経てお茶の水女子大学の教員になりました。しかしその月刊誌がなかったら、自分は大学教員になるどころか、理工学部に進学していたかさえ、わかりません。親戚が私の家に月刊誌を置き忘れた、という小さなハプニングは、実は私の人生を大きく決める非常に大きなハプニングだった、といえるかと思います。

そして縁があって2007年、出身大学でプログラミングの授業を持つことになりました。そのとき真っ先に思い出したのは、小学生のときの雑誌、そして電気屋の店頭展示品でした。私は母校に恩返しをするつもりで、そして秋葉原でお世話になったおじさんに恩返しをするつもりで、夢中になって100ページ以上にわたる講義資料を書きとどめ、そして毎年の講義に使いました。

私は2012年に本務先のお茶の水女子大学で学科長という役職に1年間就任した際に、多忙ゆえにその講義科目を後輩に譲りました。この100ページ以上の資料は現在も母校で活用されていると聞いています。

そして2013年。本務先のお茶の水女子大学にE-bookという制度ができ、書籍を無償公開できる機会を得ることができました。現在も出身大学で活用され続けている資料とはいえ、出身大学だけでなく本務先の誰かの役に立つかもしれないという思いから、またインターネット上でプログラミングを勉強する誰かの役に立つかもしれないという思いから、この拙い講義資料をE-bookで公開することにしました。

私は、小学生の時に読んだ月刊誌の連載記事を目標にして講義資料を作成しました。その方針にしたがって本書も、まずサンプルプログラムありきで構成を組み、その構成にしたがって技術的内容の説明順を決める、という考え方で構成されています。

30年前の小学生当時の私と同様な感覚で、プログラミングに親しむ学生が増えることを祈りつつ。

2013年某日
伊藤貴之

# 謝辞

本書を公開するにあたり、筆者の幾多のわがままにも丁寧にご対応いただいた、お茶の水女子大学図書・情報チームの関係各位、その他の関係者の皆様に深く感謝いたします。

本書は出版社等の専門家による校正を一切経ていない代わりに、お茶の水女子大学の大学院生による内容確認を経ています。丁寧に本書を精読してくださった、お茶の水女子大学大学院博士前期課程の福手亜弥氏、山本華子氏に深く感謝いたします。

筆者が博士号を取得する際に指導教員として大変お世話になり、さらに筆者に Java 言語の非常勤講師の機会を授けてくださった、早稲田大学大石進一教授に深く感謝いたします。

筆者の 5 年間の Java 言語の非常勤講師を通じて、授業評価アンケート等を通してたくさんの履修者の方々に参考になるお言葉を頂戴しました。当時の履修者の方々に深く感謝いたします。

筆者が初めて Java 言語に関わったのは、筆者が日本アイ・ビー・エム株式会社東京基礎研究所に研究員として勤務していた時のことです。楽しくともに研究開発を進めさせてくださった当時の研究員の方々に深く感謝いたします。

本書の執筆中に、Java 言語に関するいくつかの書籍、およびインターネット上のたくさんの情報を参考にさせて頂きました。Java 言語の普及に携わってきた多くの皆様に感謝いたします。

「まえがき」にも書きました通り、筆者がいま幸せな大学教員生活を送っているのは、小学生時代の家庭での偶然、当時の電気街での親切な計らい、それ以降の家族や友人の理解のおかげと認識しています。これまでの筆者の人生に関わってきた皆様に感謝いたします。

## 本書の構成

本書は筆者が非常勤講師を担当した大学の講義資料をそのまま再構成したものです。そのため本書の説明の難易度は全て、その大学での履修生の習熟度が基準になっています。その点をご容赦下さい。

本書は書店等で有償販売されている書籍と異なり、出版社等の専門家による校正を経ていません。講義資料をほとんどそのまま再編集した書物である点をご理解ください。

本書は大きく以下の 3 部から構成されています。全て、プログラミング実習科目を担当した時に教えた内容をそのまま載せています。

**1〜10 章:** Java 言語のサンプルプログラムおよび各文法の紹介。

**11 章:** 筆者が担当した Java 言語科目にて出題した課題。

**12〜13 章:** 筆者が担当した Java 言語科目にて資料の付録につけた各種情報。

まずは 1 章以降のサンプルプログラムを動かして頂き、続いてそのサンプルプログラムを構成する各文法を理解する、という順番で各章を理解して頂き、必要に応じて課題や各種情報をご参照頂ければと思います。

## 学習を始める前に

プログラミングを実際に始める前に、お手元のパソコン上に以下のソフトウェア環境を揃える必要があります。これらをどのように導入してどのように準備するか、どのように操作するか、については本書では記述しません。お近くの詳しい人に教わって下さい。

### コマンド実行環境

Windows ならコマンドプロンプト、Mac OS ならターミナルと呼ばれているアプリケーションです。キーボード入力でコマンドを実行するために必要ですので、使い方を覚えておいて下さい。本書では以後「ターミナル」と呼びます。

### Java 言語の開発環境

お手元のパソコンには Java Development Kit(JDK) というソフトウェアが導入されている必要があります。このあと何度となく出現する「java」および「javac」というコマンドは、この JDK を導入していることを前提として用います。

### テキストエディタ

文字情報を打ち込んでファイルに保存するアプリケーションです。Windows ならノートパッド（メモ帳)、Mac ならテキストエディットが付属されていますが、もっと高機能なテキストエディタをご自分で選ばれたほうがいいかもしれません。

### PDF ファイル閲覧環境

本書を紙ではなくパソコンの画面上で閲覧したい場合には、PDF ファイルを表示できるソフトウェアが必要になります。

# 目 次

# 1 計算

コンピュータは長い間「計算機」と訳されてきました。現在でこそコンピュータは計算以外の非常に幅広い目的で用いられていますが、コンピュータの仕組みが「計算をする機械」であることには現在も変わりありません。本章ではプログラミングの最も基礎的な技術として、計算に関するプログラミング方法について説明します。

## 1.1 【サンプルプログラム】BMI の算出

早速ですが簡単なサンプルプログラムとして、BMI を算出するプログラムを紹介します。BMI とは人間の体型を評価する値[1]で、

$$BMI = (\text{体重 } [kg])/(\text{身長 } [m])^2$$

によって算出されます[2]。
以下に示すプログラムでは、

- double 型と呼ばれる型（後述）の変数 weight に体重（67.0kg）
- 同じく、double 型の変数 height に身長（1.78m）

が、それぞれ代入されています。そして

- double 型の変数 bmi に、weight / (height * height) の算出結果

が代入されます。そして最後、「BMI=」の後に、変数 bmi に代入された値が表示されます。

```
public class Bmi {
   public static void main(String[] args) {
     double weight = 67.0;
     double height = 1.78;
     double bmi = weight / (height * height);
     System.out.println("BMI=" + bmi);
   }
}
```

それでは、上記のプログラムをテキストエディタで打ち込み、Bmi.java というファイル名で保存して下さい。そして、ターミナル上で以下の 2 個のコマンドを実行してみて下さい。

```
javac Bmi.java
java Bmi
```

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。

```
BMI=21.146319909102385
```

[1] 22 が標準値で、大きく下回ると体重過小、大きく上回ると体重過多、と言われています。
[2] 身長の単位はセンチメートルではなくメートルであることに注意。

また、変数 weight および height に別の値を代入して実行したときに、表示結果である BMI の値も変化することを、あわせて確認してください。

このような計算を実施するプログラムを、本章にて解説いたします。

## 1.2 文字列と数式

さて、本章では Java 言語を用いた計算について解説します。まずは以下のプログラムをつくり、Calculate.java というファイル名で保存して下さい。

```
public class Calculate {
   public static void main(String[] args) {
     System.out.println("3+5");
   }
}
```

まず 1 行目の public class Calculate { について説明します。w Java では **class(クラス)** をプログラムの最小単位として考えます。この 1 行目は、他のプログラムにも公開される（=public な）クラスとして、その名前を Calculate と定義しています。

続いて 2 行目の public static void main(String[] args) { ですが、細かい定義は後述するとして、Java では main と書かれたところからプログラムの実行を開始することになっています。つまり、この行はプログラム実行開始の目印を示しています。

続いて 3 行目の System.out.println(”3＋5”); ですが、これも細かい定義は後述するとして、System.out.println の次の () の中の文字列を画面に表示せよ、という処理命令が記述されています。

続いて 4 行目の } ですが、これは 2 行目の行末の { を閉じる役割をしています。5 行目の } も同様に、1 行目の行末の { を閉じる役割をしています。Java 言語や C 言語などのプログラム言語では一般的に、{ と } で囲まれたカッコ の中を、処理の一ブロックとして扱います。そしてカッコが二重以上の入れ子を構成する場合には、数式と同様に、内側のカッコどうしが対応関係をもち、外側のカッコどうしが対応関係をもちます。

では、ターミナル上で以下の 2 個のコマンドを実行してみて下さい。

```
javac Calculate.java
java Calculate
```

1 行目の javac Calculate.java という命令は、Calculate というクラスをコンパイルしなさい、という意味を持ちます。2 行目の java Calculate という命令は、Calculate というクラスの中にある main から出発してプログラムを実行しなさい、という意味を持ちます。

ここで注意すべき点は、以下の通りです。最初は紛らわしいかもしれませんので、よく理解して下さい。

- **プログラムを修正したときには javac を実行する必要があるが、修正していないときには java だけを実行すればプログラムを繰り返し実行できる。**
- **javac のあとには、クラス名ではなくファイル名を書く。よって上記の例の場合には、javac のあとは Calculate.java となる。**
- **java のあとには、ファイル名ではなくクラス名を書く。よって上記の例の場合には、java のあとは Calculate となる。**

さて、以上を実行して何が表示されましたでしょうか？ おそらく皆さんの画面では、

```
3+5
```

と表示されたことと思います。これはJava言語では、2個の¨(**引用符** または **ダブルクオーテーション**という）で囲まれた文字群を、一種の文として扱う、というように規定するからです。

このように文として扱われた文字群を、**文字列** といいます。上記のプログラムでは、「3+5」を文字列として扱います。このとき計算機は、この数式を計算させる、というようには解釈しません。

では、この引用符を消して、以下のようにプログラムを書き直して下さい。

```
public class Calculate {
   public static void main(String[] args) {
     System.out.println(3+5);
   }
}
```

そして、これを実行すると、何が表示されますでしょうか？ おそらく

```
8
```

と表示されたことと思います。これは言うまでもなく、3+5という数式を計算した結果です。

つまりJava言語では計算機は、引用符で囲まれていない数式は、「計算せよ」と命令されたものと解釈し、その計算結果を扱います。

このようにJava言語では、引用符を使うか否かによって、数式を文字列として扱うか、計算命令として扱うか、を判別します。

## 1.3 変数と代入

では今度は、Calculate.javaを以下のように書き換えてみましょう。

```
public class Calculate {
   public static void main(String[] args) {
     int i;
     i=3+5;
     System.out.println(i);
   }
}
```

そして、これを実行すると、何が表示されますでしょうか？ おそらく

```
8
```

と表示されたことと思います。つまり前節と同様に、3+5 という加算を実行した結果が表示されることでしょう。

まず 3 行目の「int i;」について説明します。この行は、i という変数を使うことを意味します。そして int は、この i という変数が整数であることを意味します。このように Java 言語（に限らず多くのプログラム言語）では、この行以降に用いられる変数の名前と性質を、予め明記する必要があります。この明記を **宣言** と呼びます。

続いて 4 行目ですが、これは右辺である 3+5 を算出した結果を i に代入しています。このように Java 言語（に限らず多くのプログラム言語）では、**右辺の算出結果を左辺の変数に代入する** という形式で、多くの算術演算を記述します。この習慣は、見慣れるまで違和感があるかもしれませんが、早く慣れて下さい。

続いて 5 行目ですが、System.out.println(i); は i の値を画面表示せよ、という意味を持ちます。4 行目の演算にて i には 3+5 の計算結果が代入されていますので、ここで 8 が画面表示される、というわけです。

ここで 5 行目を

```
System.out.println("i");
```

と書き換えて実行したら、何が表示されますでしょうか？もうおわかりですよね。ここでは

```
i
```

と表示されることと思います。i を引用符で括った場合には、この i は文として扱われ、変数としては扱われません。

## 1.4 変数の型

前節では、i という変数を用いること、そして i は整数であることを宣言する必要がある、という旨を説明しました。Java 言語（に限らず多くのプログラム言語）では、「整数」「小数」といった変数の性質を **型** と呼びます。

Java 言語での計算に必要な最小限の型の種類を、表 1 に紹介します。見ておわかりのように、同じ整数の型にも、byte, short, int, long と実に 4 種類の型があります。一般的には、計算機の規模や、計算機に処理させたい計算の規模や精度によって、型を使い分けることになります。本書では大半の場合において、整数に int を利用します。同様に、同じ小数の型にも float, double という 2 種類の型がありますが、本書では大半の場合において、小数に double を利用します。

Java 言語における変数の型について、もっと広く習得したい人は、検索エンジンで調べるか、専門書を購入して調べて下さい。

ところで、表 1 の小数の説明にて、見慣れない数値表記があるのを気がつきましたでしょうか。ここで一例として、3.40282347E+38 という表記に注目して下さい。この表記は、(3.40282347) 掛ける (10 の 38 乗) という意味を持ちます。

この表記において、3.40282347（つまり E の前まで）を**実数部** といい、+38（つまり E の後から）を**指数部** といいます。計算機の内部構造においても、一般的には小数を、実数部と指数部に分けて処理しています。この結果、計算機の内部では、小数点や有効桁数を変動させながら小数を扱うことになります。この仕組みによって計算機内部で扱われる小数を、**浮動小数点数**といいます。

表 1: Java 言語における変数の型。

| 型の名前 | 型の説明 |
|---|---|
| boolean | false(偽) または true(真) のいずれか 2 値しかとらない変数。 |
| char | Unicode という規格で定められた文字を表す変数。1 変数が 1 文字に対応する。 |
| byte | 8 ビットという単位で表現できる整数をとる変数。<br>範囲は-128〜127 |
| short | 16 ビットという単位で表現できる整数をとる変数。<br>範囲は-32768〜32767 |
| int | 32 ビットという単位で表現できる整数をとる変数。<br>範囲は-2147483648〜2147483647 |
| long | 64 ビットという単位で表現できる整数をとる変数。<br>範囲は-9223372036854775808〜9223372036854775807 |
| float | 32 ビットという単位で表現できる小数をとる変数。<br>範囲は-3.40282347E+38 〜 -1.40239846E-45<br>および+1.40239846E-45 〜 +3.40282347E+38 |
| double | 64 ビットという単位で表現できる小数をとる変数。<br>範囲は-1.7976931348623157E+308 〜 -4.94065645841246544E-324<br>および+4.94065645841246544E-324 〜 +1.7976931348623157E+308 |

表 2: Java 言語における四則演算の記号。

| 記号の種類 | 記号の説明 |
|---|---|
| a + b | a と b を加算する |
| a - b | a から b を減算する |
| a * b | a と b を乗算する |
| a / b | a を b で除算する |
| a % b | a を b で除算した剰余を求める |

## 1.5 四則演算

前節では加算（3+5）を例にして計算のプログラムを示しましたが、本節では四則演算について説明します。四則演算といえば加算（＋）、減算（−）、乗算（×）、除算（÷）を指しますが、Java 言語（に限らず多くのプログラム言語）では表 2 に示すように、四則演算の一部の記号が算数の教科書などと異なる点に注意して下さい。

# 2　条件分岐

計算機に処理させたい内容は、状況に応じて変化することがあります。「あるときは A を処理させたい」「またあるときは B を処理させたい」というように、時と場合に応じて分岐させたい、ということがあります。本章では、条件に応じて処理内容を分岐させるプログラムについて解説します。

## 2.1　【サンプルプログラム】おみくじ

早速ですが簡単なサンプルプログラムとして、「大吉」「中吉」「小吉」「凶」のいずれかをランダムに出力する、「おみくじ」のプログラムを紹介します。

このプログラムで 3 行目に用いられている Math.random() は、0.0 から 1.0 の間でランダムな値（乱数という）を生成するものです。プログラムの 3 行目では、この結果を変数 value に代入します。

続いて 4 行目以降では、その値の大小によって条件分岐します。4 行目の `if(value >= 0.75)` では、value に代入された値が 0.75 以上という条件が成立すれば、その後の { と } の間を実行します。具体的には ”Daikichi” と画面表示します。

これが成立しないときには、7 行目の else に移動して、さらに `if(value >= 0.5)` という条件分岐に入ります。ここで「value に代入された値が 0.5 以上」という条件が成立すれば、その後の { と } の間を実行します。具体的には ”Chukichi” と画面表示します。

これも成立しないときには、10 行目の else に移動して、さらに `if(value >= 0.25)` という条件分岐に入ります。ここで「value に代入された値が 0.25 以上」という条件が成立すれば、その後の { と } の間を実行します。具体的には ”Shokichi” と画面表示します。

これも成立しないときには、13 行目の else に移動して、その後の { と } の間を実行します。具体的には ”Kyo” と画面表示します。

```
public class Omikuji {
   public static void main(String[] args) {
     double value = Math.random();
     if(value >= 0.75) {
       System.out.println("Daikichi");
     }
     else if(value >= 0.5) {
       System.out.println("Chukichi");
     }
     else if(value >= 0.25) {
       System.out.println("Shokichi");
     }
     else {
       System.out.println("Kyo");
     }
   }
}
```

では早速、このプログラムを Omikuji.java というファイル名で保存して、繰り返し実行してみて下さい[3]。”Daikichi”, ”Chukichi”, ”Shokichi”, ”Kyo” のいずれかがランダムに表示されるのではないかと思います。

[3] ここでいう「繰り返し実行する」とは、javac コマンドを 1 回だけ実行して、そのあと java コマンドを繰り返す、という意味だということがおわかりでしょうか。

## 2.2 if文による条件記述

では、サンプルプログラムで多用された「if」を用いた以下のプログラムを紹介します。

```
public class If {
   public static void main(String[] args) {
     int a = 10;
     if(a % 2 == 0) {
       System.out.println("even");
     }
   }
}
```

このプログラムでは、まず 3 行目で変数 $a$ を 10 に初期化します。

続いて 4 行目の if 以下は、条件を指定します。具体的に言うと、if の次にある () の中に記述された条件が成立する時、その後にある { と } の間の命令を実行します。つまり、

もし○○であれば□□を実行する

という内容をプログラムとして記述すると、

if(○○) { □□ }

というようになるわけです。

さて、この 4 行目の () の中にある条件について解説します。左半分の a % 2 は前述の通り、$a$ を 2 で割った剰余を示します。続いてその後ですが、== 0 というのは「左辺が 0 であるかどうか」を判断する条件文を表します。つまり、$a$ を 2 で割った剰余が 0 であれば、() の中にある条件は満たされ、{ と } の間、つまり 5 行目にある System.out.println(”even”); が実行されます。

では、このプログラムを If.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。

```
even
```

では、このプログラムを以下のように書き換えて下さい。

```
public class If {
   public static void main(String[] args) {
     int a = 9;
     if(a % 2 == 1) {
       System.out.println("odd");
     }
   }
}
```

これを実行したとき、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。[4]

```
odd
```

#### 条件を記述する算術記号

さて、if 文の () の中に記述する条件として、前述のプログラムでは== という記号を使いましたが、条件を記述する算術記号は他にも数種類あります。表 3 に、その代表的な記号を列挙します。

表 3: Java 言語における条件記述。

| 算術記号 | 条件の説明 |
|---|---|
| `a == b` | a と b が等しい |
| `a != b` | a と b が等しくない |
| `a < b` | a が b より小さい |
| `a > b` | a が b より大きい |
| `a <= b` | a が b 以下（等しくてもよい） |
| `a >= b` | a が b 以上（等しくてもよい） |

### 2.3 if 文と else 文による条件分岐

続いて if に加えて else という文を加えた例を示します。else は、

もし○○であれば□□を実行する
さもなければ××を実行する

という内容を

if(○○) { □□ } else { ×× }

というように記述するために用います。
では早速、else 文を使ったプログラムの例を示します。

```
public class IfElse {
   public static void main(String[] args) {
     int a = 10;
     if(a % 2 == 0) {
       System.out.println("even");
     }
     else {
       System.out.println("odd");
     }
```

[4] even とは英語で偶数、odd とは英語で奇数を意味します。

```
    }
}
```

このプログラムの前半は、前述のプログラムと同様に、まず変数 $a$ に 10 を代入し、続いてそれを 2 で割った剰余を求め、それが 0 であるかを判定します。この条件が成立するときには、その直後の { と } の間を実行します。つまり画面には even と表示されるはずです。

一方、この条件が成立しないときには、else の直後の { と } の間を実行します。つまり画面には odd と表示されるはずです。

それでは、このプログラムを IfElse.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。

```
even
```

また、プログラムの 3 行目の代入文を、10 の代わりに奇数にしてみてください。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。

```
odd
```

## 2.4 条件の論理積と論理和

if 文の条件に、2 つ以上の条件を組み合わせて使いたい、という場合があります。Java 言語では例えば、2 つ以上の条件を同時に満たす、あるいは 2 つ以上の条件を最低一方満たす、というように条件を設定することができます。

2 つの条件を同時に満たす、つまり「△△かつ▽▽」という条件を記述するには、&&という記号を用いて、

if(△△ && ▽▽)

というように記述します。

2 つの条件の最低一方を満たす、つまり「△△または▽▽」という条件を記述するには、|| という記号を用いて、

if(△△ || ▽▽)

というように記述します。

ここで「△△かつ▽▽」というように、複数の条件の両方、という記述を**論理積**といいます。また「△△または▽▽」というように、複数の条件の一方、という記述を**論理和**といいます。

上述の && および || のように、論理積や論理和などの演算を記述する記号を、**論理演算子**といいます。論理演算子の代表的なものを、表 4 に列挙します。

表 4: Java 言語における論理演算子。

| 記号の種類 | 記号の説明 |
|---|---|
| && | △△かつ▽▽ |
| \|\| | △△または▽▽ |

論理演算子を用いたプログラムの例を、以下に示します。このプログラムでは4行目のif文にて、a % 2 == 0 および $a >= 5$ の両方の条件を満たした場合に限り、OKと表示します。どちらか片方でも満たさなければ、NGと表示します。

```
public class IfElse {
   public static void main(String[] args) {
     int a = 10;
     if(a % 2 == 0 && a >= 5) {
       System.out.println("OK");
     }
     else {
       System.out.println("NG");
     }
   }
}
```

このプログラムでは論理積の記号 && を例にしましたが、論理和の記号 || も、用法は全く同じです。

# 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]);
     }
   }
}
```

# 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。

# 5　プログラミングの習慣

ここまで多くのプログラムを書いてきましたが、そろそろプログラムの行数も多くなり、自分が書いてきたプログラムを自分で読み返しても理解しにくい、という状況になり始めていることと思います。

プログラミングには、プログラムを後から自分で読み返したとき、あるいはプログラムを他の人に渡したとき、などのために便利な習慣や規則がいくつか伝承されています。ここで、世界中の多くのプログラマが倣っている、プログラミングのいくつかの習慣や規則について紹介します。

## 5.1　インデント

いままでに紹介したプログラムでは、命令や構文 1 個ごとに改行をしましたし、また特定の行では行頭に所定の空白文字を加えていました。

しかし実をいうと、Java 言語や C 言語を含むいくつかのプログラム言語では、改行や行頭空白文字はプログラムに影響を及ぼしません。

極端な例として、行頭空白文字を全部消したプログラムを書いてみます。

```
public class Hello {
public static void main(String[] args) {
System.out.println("Hello, world!");
}
}
```

これで javac コマンドと java コマンドを使ってみてください。どうですか？ 普通に Hello, world ! と表示されますよね？

今度はさらに、改行も全部消してみましょう。（本書では紙面の大きさ都合により、途中で折り返されて表示されるかもしれません。）

```
public class Hello { public static void main(String[] args) { System.out.println("Hello,
world!"); } }
```

これで javac コマンドと java コマンドを使ってみてください。どうですか？ 普通に Hello, world ! と表示されますよね？

しかし実際には、ほとんどのプログラマは、今までに本書で出てきたのと同様に、プログラムに改行や行頭空白文字を加えています。それは、ほとんどのプログラマが、この書き方が「読みやすい」と考えているからであり、また「世界中のプログラマが同じようなマナーでプログラムを書いたほうが便利」と考えているからです。

今後のプログラムの書き方の目安として、

- 命令や代入文 1 個ごとに改行する。
- 中カッコを開く（{）たびに、行頭空白文字の数を増やす。
- 中カッコを閉じる（}）たびに、行頭空白文字の数を元に戻す。

ということを習慣化してください。

この行頭空白文字を **インデント**または **字下げ** と呼ぶことがあります。

なお、ここで気をつけて欲しいのは、

**日本語入力時の全角空白文字を、インデントに用いることはできない**

という点です。くれぐれも忘れないで下さい。

## 5.2 カッコの省略

if 文、for 文、while 文、do-while 文などに付随して実行される処理を囲む中カッコ（{ と }）は、カッコ内の処理が 1 文（セミコロン 1 個）だけのときに省略することができます。例えば

```
if(a >= 1) {
   System.out.println(a);
}
```

は

```
if(a >= 1)
   System.out.println(a);
```

というように書き換えることができます。ただし本書では、このようなカッコの省略は行いません。

## 5.3 コメント

プログラムを構成する各々の処理や命令の意味をわかりやすくするために、多くのプログラマは**コメント**を記入します。

コメントは計算機がコンパイル時に読み飛ばすものであり、プログラマを含む人間が画面などの上で読むだけのために記載されます。

コメントの書き方は、以下の 2 種類です。

- /* と */ で囲まれた間にコメントを書く。この間に改行があってもよい。
- // から後、その行だけにコメントを書く。

以下に、コメントや空行を意図的に多く加えたプログラムの例を書きます。本書では以下、日本語でコメントを記入することにします。

```
// BMI 値を計算するクラス
public class BmiMethod {

   // 体重と身長を受け取り、BMI 値を算出して返すメソッド
   public static double calculateBmi(double weight, double height) {
     // BMI の公式は (体重)/(身長^2)
     double result = weight / (height * height);
     return result;
   }

   // メインメソッド
```

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

        double w = 67.0; // 体重 67.0kg
        double h = 1.78; // 身長 1.78m

        // BMI 値を算出して、変数 bmi に代入する
        double bmi = calculateBmi(w, h);

        // BMI 値を画面に出力する
        System.out.println("BMI=" + bmi);
    }
}
```

## 5.4 変数名・定数名・クラス名・メソッド名

プログラムに出てくる変数、定数、クラス、メソッドなどの名前にも一定の習慣があり、多くのプログラマがそれに倣ってプログラムを書いています。本書では以下の習慣によって、変数、定数、クラス、メソッドの名前をつけています。参考になれば幸いです。

**変数名** 小文字だけで記述する。

(例) k, m, n, count, key, light

**定数名** 大文字だけで記述する。2 つ以上の単語を合成した定数名をつくる場合には、単語と単語の間をアンダースコアで接続する。

(例) COUNT, KEY, ONE_TIME, COLOR_RED

**メソッド名** 原則として小文字だけで記述する。ただし複数の単語を組み合わせて名前をつくる場合には、最初の単語は小文字だけで記述し、2 つめ以降の単語は最初の文字だけを大文字にする。

(例) get(), set(), initialize(), getAnswer(), setValue() なお 2 つ以上の単語を用いるときの習慣として、「動詞＋目的語」という単語の組み合わせが非常に多い。

**クラス名** 最初の文字だけを大文字にする。複数の単語を組み合わせて名前をつくる場合には、各単語の最初の文字だけを大文字にする。

(例) Color, Shape, RedTriangle, FirstGrade

そのクラスが一定の処理を担当するときには、「…の処理をする人」という意味をこめて -er で終わる接尾語を使うことが多い。

(例) Calculator, ColorInitializer, MessageReceiver

# 6　入力と出力

1.1 節に示した BMI 算出のサンプルプログラムでは、体重や身長の値はプログラムに直接書き込まれていました。しかしこれだと、他の人の BMI を算出する際には、わざわざプログラムを書き換える必要があります。プログラムの実行開始時、あるいはプログラムの実行途中に、利用者が値をキーボード入力できれば、プログラムはもっと便利なものになるに違いありません。

本章では、キーボードなどを通して利用者が値を入力する仕組みとして、実行時の引数、標準入力、について解説します。

続いて標準入力と対称な関係にある標準出力について解説します。標準出力とは、計算機がディスプレイなどに値を出力する仕組みのことです。

## 6.1　実行時の引数

1.1 節に示した BMI 算出のサンプルプログラムを、以下のように書き換えてみて下さい。

```
public class Bmi {
   public static void main(String[] args) {
     double weight = Double.parseDouble(args[0]);
     double height = Double.parseDouble(args[1]);
     double bmi = weight / (height * height);
     System.out.println("BMI=" + bmi);
   }
}
```

そしてこれを、いままでと違って、以下のように実行してみて下さい。

```
javac Bmi.java
java Bmi 67.0 1.78
```

どうでしょうか？ 以下のような実行結果が表示されましたでしょうか。

```
BMI=21.146319909102385
```

今回のプログラムでは、Bmi.java ファイルには体重や身長の具体的な値は記入されていません。そのかわりに実行時に、「java Bmi 67.0 1.78」と、体重や身長の値を添えて実行することで、BMI の計算を実現しています。このように、実行時に体重や身長の値を書き添えることができれば、他の人の BMI を算出するために、わざわざプログラムを書き換える必要がないので、1.1 節で示したプログラムより実用的である、と考えることができるでしょう。以下、このプログラムについて説明しましょう。

### main メソッドに引き渡される実行時の引数

まず、今までにも使われてきた 2 行目の public static void main(String[] args) ですが、main の後のカッコの中にある「String[] args」も、実は変数の型と名前を宣言しています。String というのは文字列を表すクラスのことです（4.2 節参照）。その後の [] は配列を表します（3.5 節参照）。そして、args が文字列を表す変数の名前になります。

続いて 3 行目と 4 行目ですが、args[0] と args[1] はそれぞれ、配列となっている変数 args の 0 番目と 1 番目の値に相当します。ここで 0 番目および 1 番目の値はそれぞれ、実行時に「java クラス名」の後に書き添えられた文字列、つまり 67.0 および 1.78、ということになります。

#### 文字列から実数への変換

上記の「67.0」および「1.78」という文字列は、あくまでも文字列として扱われていて、この時点で計算機は 67.0 および 1.78 を実数だと思っていません。そこで BMI を計算するためには、これらの文字列を実数に変換します。3 行目の Double.parseDouble(args[0]) は、args[0] に代入されている 67.0 という文字列を、double 型の実数に変換する働きを持っています。4 行目の Double.parseDouble(args[1]) も同様です。[7]

## 6.2 標準入力

前節で説明した「実行時の引数」は、プログラムの実行開始時に、変数の値を指定できるようにする、というものでした。

しかし実際には、プログラムの実行開始時だけでなく、プログラム実行中の任意のタイミングで変数の値をキーボード入力できると、さらに便利なプログラムを開発することができます。本節では、プログラムの途中でのキーボード入力の方法について説明します。

1.1 節に示した BMI 算出のサンプルプログラムを、BmiKeyboard.java というファイルにコピーして、以下のように書き換えてみて下さい。

```
import java.io.*;

public class BmiKeyboard {
   public static void main(String[] args) {
     double weight = 67.0;
     double height = 1.78;
     try {
       InputStreamReader isr = new InputStreamReader(System.in);
       BufferedReader br = new BufferedReader(isr);
       System.out.println("Please input the weight");
       weight = Double.parseDouble(br.readLine());
       System.out.println("Please input the height");
       height = Double.parseDouble(br.readLine());
       br.close();
     }
     catch (Exception e) {
       System.out.println(e);
     }
     double bmi = weight / (height * height);
     System.out.println("BMI=" + bmi);
   }
```

[7] 参考までに、この Double.parseDouble() は、Double というクラスの中にある parseDouble() というメソッドを呼び出す、ということを意味しています。

}

このプログラムでは、キーボード入力によって体重と身長の値を受けつけ、その値をもとに BMI 値を算出します。なお、多くのオペレーティングシステムでは、キーボードによる文字入力を**標準入力**と呼んでいます。

このプログラムの内容を順に追っていきましょう。

まず 1 行目の import については、7.2 章で後述します。

3,4 行目では、変数 weight および height を宣言し、値を代入しています。

5 行目の try については、7.3 章にて後述します。現時点で注釈しておきたい点として**BufferedReader クラスの readLine() メソッドおよび close() メソッドは、try の後の { と } の間に記述するべきである、**という点をあげておきます。

6 行目の InputStreamReader クラスの変数 isr は、標準入力の情報を文字として扱うために用います。7 行目の BufferedReader クラスの変数 br は、その文字情報を、例えば 1 行単位といったまとまった情報として扱うために用います。

8 行目の System... 行では、Please input the weight という文字列を表示します。これが表示されたら、体重の値をキーボード入力して Enter を押してください。9 行目の br.readLine() というメソッドで、そのキーボード入力した 1 行ぶんの文字情報を抽出します。9 行目の Double.parseDouble() は、その文字情報を double 型の実数に変換します。この値が、変数 weight に代入されます。

10 行目の System... 行では、Please input the height という文字列を表示します。これが表示されたら、身長の値をキーボード入力して Enter を押してください。11 行目の br.readLine() というメソッドで、そのキーボード入力した 1 行ぶんの文字情報を抽出します。11 行目の Double.parseDouble() は、その文字情報を double 型の実数に変換します。この値が、変数 height に代入されます。

12 行目の br.close() は、変数 br による文字情報入力を終了するときに、必ず一度だけ呼び出す必要があります。

14 行目の catch(Exception e) および 15 行目の System.out.println(e) については、7.3 章で後述します。現時点で注釈しておきたい点として、**try の } の直後に必ず catch 文を入れること、**という点をあげておきます。

そして最後に、17 行目で BMI 値を算出して変数 bmi に代入し、18 行目でその値を画面に出力します。

では、このプログラムを実行してみてください。そして、いままで説明したとおりに実行できることを確認してください。

## 6.3 標準出力

前節で説明した「標準入力」に対称な概念として、**標準出力**があります。多くのオペレーティングシステムでは、ディスプレイ上での文字出力を標準出力と位置づけています。

いままで何度も出てきた System.out.println() は、文字列を標準出力するメソッドです。具体的にいうと、System クラスの中に out 変数があり、out 変数が println メソッドを有している、という関係になります。

標準出力に関する知識は、他にもいろいろあるのですが、本書では割愛します。

## 6.4 【サンプルプログラム】相性占い

では、標準入力を用いたサンプルプログラムを紹介します。このプログラムは、人物 A の誕生日と、人物 B の誕生日を入力し、この 2 人の相性を 0%から 100%の間の値で算出するものです。[8]

[8] プログラム中のコメントにも書いてあるように、この相性算出式は当てずっぽうですので、算出された値に対して責任を持つことはできませんが、ご了承ください。

このプログラムを Uranai.java というファイル名で保存して、実行してください。そして、以下の順番に誕生日を入力してください。

1. Month of birthday A と表示されたら、人物 A の誕生月を入力する。
2. Day of birthday A と表示されたら、人物 A の誕生日を入力する。
3. Month of birthday B と表示されたら、人物 B の誕生月を入力する。
4. Day of birthday B と表示されたら、人物 B の誕生日を入力する。

例えば、1 月 3 日生まれの人と、4 月 12 日生まれの人の相性を計算するときは、1, 3, 4, 12 の順に 4 つの数字を入力してください。そうすると、0 から 100 の間の数字が出力されます。これが相性をパーセントで表現したものになります。

```
import java.io.*;

// 相性占いのクラス
public class Uranai {

   // 元旦からの合計日数を求める
   static int calculateTotalDays(int month1, int day1) {
     int total = 0;
     int days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

     // 誕生月の前の月までの合計日数を求める
     for(int i = 0; i < (month1 - 1); i++) {
       total += days[i];
     }

     // 合計日数に日を足す
     total += day1;

     // 合計日数を返す
     return total;
   }

   // 相性をパーセントで求める
   static int calculateChemistry(int total1, int total2) {
     int chemistry;

     // 相性の算出式 （当てずっぽうな算出式なので信頼できるものではない）
     chemistry = (total1 + total2) % 101;

     // 相性を返す
     return chemistry;
   }
```

```
  // メインメソッド
  public static void main(String[] args) {

    // 2 人の誕生日の月と日を代入する変数
    int month1 = 0, month2 = 0, day1 = 0, day2 = 0;

    // 人物 A の月、日、人物 B の月、日、の順に標準入力する
    try {
      InputStreamReader isr = new InputStreamReader(System.in);
      BufferedReader br = new BufferedReader(isr);
      System.out.println("Month of birthday A");
      month1 = Integer.parseInt(br.readLine());
      System.out.println("Day of birthday A");
      day1 = Integer.parseInt(br.readLine());
      System.out.println("Month of birthday B");
      month2 = Integer.parseInt(br.readLine());
      System.out.println("Day of birthday B");
      day2 = Integer.parseInt(br.readLine());
      br.close();
    }
    catch (Exception e) {
      System.out.println(e);
    }

    // 人物 A の誕生日の元旦からの合計日数を求める
    int total1 = calculateTotalDays(month1, day1);
    // 人物 B の誕生日の元旦からの合計日数を求める
    int total2 = calculateTotalDays(month2, day2);
    // 人物 A と人物 B の相性をパーセントで求める
    int chemistry = calculateChemistry(total1, total2);
    // 人物 A と人物 B の相性を標準出力する
    System.out.println("chemistry=" + chemistry);
  }
}
```

なお、このプログラムで注意すべき新しい点を、いくつか説明します。

#### 初期値を指定する配列の宣言方法

calculateTotalDays メソッドにて、days という配列の変数が使われています。本来なら 3.5 節で説明したとおり、配列の変数を宣言するときには、

```
int days[] = new int[12];
```

というような宣言が必要です。しかし、配列を構成する各々の値がすでに確定しているときには、

```
int days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

と記載することで、適切な個数の配列を確保し、その各々の値を代入してくれます。

#### 複数の変数の 1 行での宣言

main メソッドにて、以下の 4 つの int 型の変数が宣言されています。

```
int month1 = 0, month2 = 0, day1 = 0, day2 = 0;
```

変数の宣言においては上記のように、複数の変数をカンマ (,) で区切って、1 行につなげて記載することが許されています。

#### 文字列から整数への変換

6.1 節にて「文字列から実数への変換」について説明しましたが、これと同様に「文字列から整数への変換」を施すこともできます。main メソッドには Integer.parseInt(br.readLine()) という処理が 4 回呼び出されていますが、これは br.readLine() によって抽出された文字列を整数に変換するために行われています。

#### メソッドによる処理の小分け

main メソッドでは、引数を入れ替えて 2 回、calculateTotalDays メソッドを呼び出しています。**このように、複数回にわたって同じ処理をする場合、しかもそれが異なる変数を用いる場合には、その処理部分をメソッドとして分けることで、プログラムを読みやすく、また保守しやすくします。** 特に大きなプログラムを書くときには、積極的にメソッドをつくって処理を小分けにする、ということを心がけましょう。

# 7　その他の知識：パッケージ・例外処理・いくつかのキーワード

## 7.1　package

Java 言語では、1 つのクラスに対して 1 つのファイル、という体制で開発を進めます。大きなプログラムでは、クラスが数百個、という状況も発生します。これを全て一括管理するのは大変なので、クラスをいくつかのグループに分類して保管する仕組みが必要になります。それはちょうど、たくさんのファイルを、ディレクトリで分類して保管するのと同じような仕組みに相当します。

### パッケージとは

Java 言語では、**パッケージ**という考え方により、クラスを分類することができます。

例えば、いくつかのクラスを、a という名前のパッケージに格納したとします。このとき、a に格納されたクラスの 1 行目には、

```
package a;
```

というように記載する必要があります。

ディレクトリと同様に、パッケージも二重三重に作成することが可能です。例えば a というパッケージの中に b というパッケージを作成したとします。このとき、親パッケージを a とし、子パッケージを b とすると、a と b をピリオドで連結して a.b というように記載します。このような場合には、a.b に所属するクラスの 1 行目には、

```
package a.b;
```

というように記載する必要があります。

### パッケージとディレクトリの関係

Java 言語では、自分でパッケージをつくって、自分でつくったクラスを管理するときには、パッケージとディレクトリに対応関係がとれていないといけません。このときの対応関係について、以下の 3 つのルールがあります。

**ルール 1:**　パッケージに属するクラスのファイルは、そのパッケージの構造と同じ構造のディレクトリに属する必要があります。例えば a というパッケージに属するクラスのファイルは、a というディレクトリを作り、その中に置かなければなりません。例えば a.b というパッケージに属するクラスのファイルは、a というディレクトリの中に b というディレクトリを作り、その中に置かなければなりません。

**ルール 2:**　パッケージに属するクラスを実行するには、パッケージに対応するディレクトリの上のディレクトリから実行しなければならない。例えば program というディレクトリに a というディレクトリを作り、a というパッケージに属するクラスのファイルを置いたときには、java 命令は program というディレクトリから実行しなければならない。

**ルール 3:**　java 命令では、パッケージ名をつけてクラス名を表現する必要がある。例えば a.b というパッケージに属する Hello というクラスを実行するときは、

```
java a.b.Hello
```
というように実行する必要がある。

## 7.2 import

Java 言語では、同一のパッケージに属するクラスどうし、あるいはどのパッケージにも属さないクラスどうしは、実際に同じディレクトリにファイルが置いてあれば、クラス名を指定するだけで互いに参照できます。

例えば 4.4 節で紹介した BmiMethod1 クラスは、同じディレクトリに置いてある BmiMethod2 クラスを、クラス名を指定するだけで参照できています。

しかし異なるパッケージに属するクラスを参照する際には、そのクラスがどのパッケージに属するかを含めて指定しないといけません。

例えば 4.4 節で紹介した BmiMethod2 クラスが、a.b というパッケージに属していたとします。この BmiMethod2 クラスを BmiMethod1 クラスから参照する際には、a.b.BmiMethod2 というように、パッケージ名とクラス名をピリオドで連結して表記しないといけません。

でも、毎回いちいちパッケージ名を記載するのは面倒くさいので、最初にあらかじめ、そのクラスがどのパッケージに属するものであるかを宣言する構文が生まれました。それが **import** 文です。

### import 文の記述方法

一例として、java.io というパッケージの中に入っている InputStreamReader というクラスを利用する可能性があるときには、プログラムの冒頭に

```
import java.io.InputStreamReader;
```

と記述します。この記述によって InputStreamReader というクラスは、それ以降はパッケージ名を指定しなくても使えるようになります。

また別の import 文の例として、6.2 節に紹介したものを説明します。

```
import java.io.*;
```

この記述は、java.io というパッケージに属する全てのクラスを利用する、ということを意味します。ここで「*」は「全ての」という意味で利用されます。このような表現を、**正規表現**といいます。

6.2 節のプログラムでは、InputStreamReader および BufferedReader というクラスが使われていますが、これらはいずれも java.io というパッケージに属するものです。

### 最初から使えるパッケージ

ところで、java.io というパッケージは、皆さんが自分で作った覚えはないだろうと思います。このように Java 言語では、皆さんが自分で作っていなくても最初から使えるパッケージが、いくつか用意されています。本資料では java.io の他に、8.1 節のプログラムにて、java.util というパッケージが import されています。

さらには、import しなくても最初から使えるパッケージとして、java.lang というパッケージが用意されています。典型的な例として、これまでに何度も出てきた以下のクラスは、java.lang パッケージの中にありますので、import 文を一切書かなくても最初からクラス名を指定するだけで利用できます。

表 8: java.lang パッケージに属するクラスのうち本資料で用いるもの。

| クラスの名前 | クラスの説明 |
|---|---|
| String | 文字列を扱うクラス。 |
| Integer | 整数に関する処理のためのクラス。 |
| | 本資料では Integer.parseInt() などを用いる。 |
| Double | 浮動小数点数に関する処理のためのクラス。 |
| | 本資料では Double.parseDouble() などを用いる。 |
| Math | 数式演算に関するクラス。 |
| | 本資料では Math.rand() などを用いる。 |
| System | コンピュータシステムに関わるクラス。 |
| | 本資料では System.out.println() などを用いる。 |

## 7.3 例外処理：try と catch

プログラムを実行しているときに、いろいろな例外が起こることがあります。例えば「整数をゼロで割ろうとした」「配列の大きさ以上の添え字を使おうとした」といったものが典型的な例外です。一般的にプログラムは、例外が起こると実行を停止してしまいます。

Java 言語では、例外が起こったときに、実行を停止しない代わりに、その例外から正常な状況に戻るための処理をさせる、というプログラミングが可能です。このような処理を **例外処理** といいます。

例外処理のための構文に、**try - catch 文** があります。 この文では、

○○という処理中に、△△という例外が発生したら、□□を実行する

という処理を、

try { ○○ } catch( △△ ) { □□ }

というように記述します。

6.2節のプログラムを例にすると、try の後の { から } までの間で例外が発生すると、その後の catch(Exception e) にて Exception クラスの変数 e に例外の内容が記録されます。そしてその後の System.out.println(e); にて、例外の内容が標準出力されます。

```
...
     try {
       InputStreamReader isr = new InputStreamReader(System.in);
...
     }
     catch (Exception e) {
       System.out.println(e);
     }
...
```

#### 例外処理が必須である場合・必須でない場合

例外には「例外処理を記述しなくてもいいもの」と「例外処理を記述しなくてはいけないもの」があります。

一例として、以下のようなメソッドは、例外処理を記述しなくてはなりません。

- BufferedReader クラスの readLine() メソッド、close() メソッド（6.2 節参照）
- BufferedWriter クラスの write() メソッド、flush() メソッド、newLine() メソッド（8.2 節参照）

このようなメソッドを実行する部分は、try 文に続いて { と } で囲む必要があります。

## 7.4 public と private

Java 言語では、クラス、メソッド、変数の「公開度」を設定することができます。この公開度とは、異なるクラスやパッケージから、そのクラス、メソッド、変数がアクセス可能であるかどうか、を示すものです。

この設定のために Java 言語では、**アクセス修飾子**という用語を使います。ここではその代表例として、publuc と private を紹介します。

#### public

いままで多くのプログラムにおいて、多くの場面で public という単語が使われてきましたが、長らく説明を省略していました。

public という単語は、クラスやメソッドの名前の接頭に用いられてきました。これらとは別に、クラスやメソッドだけでなく、変数の名前の接頭にも用いることができます。

表 9: public を接頭に用いる例。

| | |
|---|---|
| クラス | public class Bmi |
| メソッド | public static void main(String args[]) |
| 変数 | public int i = 1; |

public とは、そのクラス、メソッド、変数が、たとえ違うパッケージからでも自由に呼び出せることを表しています。ただし、変数に public を指定することは、あまり推奨されていません。

#### private

public の代わりに private をつけると、メソッドや変数を「そのクラス内からしか呼び出せない」という状態になります。private は、クラス外部に対して最も隠蔽された状態を表します。

#### public も private も指定しなければ

public も private も指定していないメソッドや変数は「同じパッケージ内の別のクラスからのみ呼び出せる」という状態になります。

## 7.5 static

いままで説明してきたプログラムでは、クラスを利用する際には、new 命令などを用いてインスタンスを生成してきました。例えば 4.4 節で紹介したプログラムでは、BmiMethod2 クラスを使用する際に、

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

というように、new 命令によって BmiMethod2 クラスのインスタンスを生成し、それを変数 method2 に代入していました。

これに対して、インスタンスを生成しないで変数やメソッドを利用する方法もあります。この場合には **static** という修飾子を使う必要があります。

一例として、4.4 節で紹介した BmiMethod1 クラスを、以下のように書き換えてみてください。

```
public class BmiMethod1 {
   public static void main(String[] args) {
     BmiMethod2.weight = 80.0;
     BmiMethod2.height = 1.80;
     double bmi = BmiMethod2.calculateBmi();
     System.out.println("BMI=" + bmi);
   }
}
```

さらに BmiMethod2 クラスを、以下のように書き換えてみてください。

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

そして、この 2 つのクラスをコンパイルしてください。

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

そして、main メソッドのある BmiMethod1 クラスを実行してください。

```
java BmiMethod1
```

どうでしょうか？ ちゃんと実行できましたでしょうか？

### 実は今までにも使われている static

実はいままでにも本資料では、static を使って定義された多くのメソッドを使っています。例えば、

```
Integer.parseInt();
Double.parseDouble();
```

はそれぞれ、Integer クラス、Double クラスの中で static を使って定義された parseInt() メソッド、parseDouble() メソッドを表しています。また、

```
System.out.println();
```

は、System クラスの中に static を使って定義された out という変数があり、その中で static を使って定義された println() というメソッドがある、ということを表しています。

# 8　ファイル入出力

6.2節では、プログラムの実行途中で、利用者がキーボードを用いて値を入力する方法について説明しました。しかし、プログラムが必要とする値が非常に多い場合、それを全てキーボード入力するのは手間が大きいでしょう。

また6.3節では、プログラムに用いられる値をディスプレイに出力する方法について説明しました。しかし、プログラムが出力する値が、ディスプレイに入りきらないくらい多い場合、それを全て表示して読むのは難しいでしょう。さらに、その出力した値を明日もう一度使いたい、というときにもディスプレイにしか出力できないのでは不便でしょう。

これらの問題を解決するために、本章ではファイルを操作する方法について解説します。

## 8.1　ファイル入力

本節では、プログラムが別のファイルの内容を読み込む処理について解説します。なお、Java言語を含む多くの言語では、ファイルの内容を読み込む処理を**ファイル入力**と呼びます。

まずテキストエディタでwh.txtというファイルを新規作成し、その中身を以下のように記述してください。

```
57.0   1.64
59.0   1.66
64.0   1.77
70.0   1.73
79.0   1.81
41.0   1.49
45.0   1.55
47.0   1.59
62.0   1.59
58.0   1.67
```

このファイルは、10人の体重（単位kg）と身長（単位m）を記述したもので、各行の左側に体重、右側に身長が記載されているものとします。

では、このファイルの内容をプログラムで読み込み、10人ぶんのBMIを一気に計算して表示するプログラムを紹介します。

まず以下のコマンドで、BmiMethod.java を BmiFileIO.java というファイルにコピーし、続いてそれをテキストエディタで表示してください。そして BmiFileIO.java の中身を、以下のように書き換えてください。

```
import java.io.File;
import java.io.FileReader;
import java.io.BufferedReader;
import java.util.StringTokenizer;

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

```
    return result;
  }

  public static void main(String[] args) {
    double weight = 67.0;
    double height = 1.78;
    String filename = "wh.txt";
    File file = new File(filename);
    try {
      FileReader freader = new FileReader(file);
      BufferedReader breader = new BufferedReader(freader);
      while(breader.ready()) {
        String line = breader.readLine();
        if(line == null) break;
        StringTokenizer token = new StringTokenizer(line);
        if(token.countTokens() < 2) continue;
        String w = token.nextToken();
        String h = token.nextToken();
        weight = Double.parseDouble(w);
        height = Double.parseDouble(h);
        double bmi = calculateBmi(weight, height);
        System.out.println("w=" + weight + " h=" + height + " BMI=" + bmi);
      }
      breader.close();
    } catch(Exception e) {
      e.printStackTrace();
    }
  }
}
```

では、このプログラムの中身について説明しましょう。

まず1〜4行目では、BmiFileIO クラスで用いる別のクラスとして、File, FileReader, BufferedReader, StringTokenizer クラスを import しています。ここでは明示的に 4 つのクラスの名前を記述していますが、7.2 節で説明したように、

```
import java.io.*;
import java.util.*;
```

というように記述しても構いません。

calculateBmi メソッドについては、4.3 節で紹介した BmiMethod クラスと同じですので、ここでは省略します。

続いて 16 行目に、**File** クラスが新しく出てきます。File クラスとは、特定のファイルを扱うためのクラスです。ここでは変数 filename に代入されている、wh.txt というファイル名に対して、File クラスのインスタンスを生成し、これを file という変数に代入します。

続いて 18 行目に、**FileReader** クラスが新しく出てきます。FileReader クラスは、特定のファイルの内容を読むためのクラスです。ここでは変数 file に代入されているファイルに対して、FileReader クラスのインスタンスを生成し、これを freader という変数に代入します。

続いて 19 行目に、BufferedReader クラスが出てきます。BufferedReader クラスの使い方は、6.2 節で紹介した通りです。このクラスは標準入力だけでなく、ファイル入力でも同様に扱えます。

続いて 20 行目に、breader.ready() というメソッドが出てきます。この行では、変数 breader が読み込み可能である状態を確認していて、読み込み可能である限り while 文による反復を実行する、という意味を持ちます。

続いて 21 行目にて、breader.readLine() というメソッドにより、ファイルの内容を 1 行読み、この結果を変数 line に代入します。この結果が null であるときには、ファイルの内容を最後まで読んだと判断し、break 文により while 文の反復を終了します。

続いて 23 行目にて、**StringTokenizer** クラスが新しく出てきます。このクラスは文字列を所定の文字で区切るものです。この場合には、変数 line に代入された文字列を、空白文字で区切ります。例えば、wh.txt の 1 行目

```
57.0 1.64
```

を、57.0 および 1.64 の 2 つの文字列に区切ります。この区切った結果は、nextToken() というメソッドによって、以下のように得ることができます。

- 1 回目の token.nextToken() メソッドの結果として、空白文字で区切った左側（この場合 57.0）を返し、これを String クラスの変数 w に代入します。
- 2 回目の token.nextToken() メソッドの結果として、空白文字で区切った右側（この場合 1.64）を返し、これを String クラスの変数 h に代入します。

ただし注意しないといけない点として、2 つの文字列に分割できない行が混じっていたときには、token.nextToken() メソッドを呼び出すとエラーになってしまいます。これを避けるために、24 行目に if(token.countTokens() < 2) continue; という行が混じっています。これは、2 つ以上の文字列に区切れなければ continue する、という意味です。

続いて 27,28 行目にて、Double.parseDouble() というメソッドが出てきます。このメソッドの意味については 6.1 節を参照してください。

29 行目の calculateBmi() メソッド、および 30 行目の System.out.println() メソッドについては、今まで何度も出てきたので省略します。32 行目の catch 以下についても省略します。

さて、このプログラムを実行してみましょう。以下のように表示されましたでしょうか？

```
w=57.0 h=1.64 BMI=21.192742415229034
w=59.0 h=1.66 BMI=21.410944984758313
w=64.0 h=1.77 BMI=20.428357113217785
w=70.0 h=1.73 BMI=23.38868655818771
w=79.0 h=1.81 BMI=24.114038033027075
w=41.0 h=1.49 BMI=18.467636592946263
w=45.0 h=1.55 BMI=18.730489073881373
w=47.0 h=1.59 BMI=18.59103674696412
w=62.0 h=1.59 BMI=24.524346347059055
w=58.0 h=1.67 BMI=20.796729893506402
```

## 8.2 ファイル出力

本節では、プログラムが別のファイルに内容を書き出す処理について解説します。なお、Java 言語を含む多くの言語では、ファイルに内容を書き出す処理を**ファイル出力** と呼びます。

ここで、先ほどの BmiFileIO.java を改良して、画面出力している BMI 値を bmi.txt というファイルに出力する、ということを考えます。

では BmiFileIO.java の中身を、以下のように書き換えてください。

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.StringTokenizer;

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

   public static void main(String[] args) {
     double weight = 67.0;
     double height = 1.78;
     String filename = "wh.txt";
     String filename2 = "bmi.txt";
     File file = new File(filename);
     File file2 = new File(filename2);
     try {
       FileReader freader = new FileReader(file);
       BufferedReader breader = new BufferedReader(freader);
       FileWriter fwriter = new FileWriter(file2);
       BufferedWriter bwriter = new BufferedWriter(fwriter);
       while(breader.ready()) {
         String line = breader.readLine();
         if(line == null) break;
         StringTokenizer token = new StringTokenizer(line);
         if(token.countTokens() < 2) continue;
         String w = token.nextToken();
         String h = token.nextToken();
         weight = Double.parseDouble(w);
         height = Double.parseDouble(h);
         double bmi = calculateBmi(weight, height);
         String line2 = "w=" + weight + " h=" + height + " BMI=" + bmi;
         bwriter.write(line2, 0, line2.length());
         bwriter.flush();
```

```
          bwriter.newLine();
        }
        breader.close();
        bwriter.close();
      } catch(Exception e) {
        e.printStackTrace();
      }
    }
}
```

では、このプログラムの中身について説明しましょう。

まず 1〜6 行目では、BmiFileIO クラスで用いる別のクラスとして、File, FileReader, FileWriter, BufferedReader, BufferedWriter, StringTokenizer クラスを import しています。

19,20 行目に、**File** クラスが 2 回出てきます。1 回目は、変数 filename に代入された wh.txt というファイルのインスタンスを、変数 file に代入します。2 回目は、変数 filename2 に代入された bmi.txt というファイルのインスタンスを、変数 file2 に代入します。前者は読み込むファイル、後者は書き出すファイルですが、ここでは両者を区別せずに、同様に new 命令でインスタンスを生成します。

続いて 24 行目に、**FileWriter** クラスが新しく出てきます。FileWriter クラスは、特定のファイルに内容を書くためのクラスです。ここでは変数 file2 に代入されているファイルに対して、FileWriter クラスのインスタンスを生成し、これを fwriter という変数に代入します。

続いて 25 行目に、**BufferedWriter** クラスが新しく出てきます。BufferedWriter クラスは、標準出力およびファイル出力において、内容を 1 行単位で書き出すなどの機能を有するクラスです。

続いて 36 行目に、String line2= で始まる代入文があります。これは、引用符で囲まれた w=,h=,BMI= といった文字列や、変数 weight, height, bmi の値を、一続きに並べてできる文字列を、変数 line2 に代入する、というものです。この結果として変数 line2 では、BmiFileIO.java が標準出力していたときと同じ内容の文字列を格納することになります。

続いて 37〜39 行目にて、変数 bwriter のメソッドが続々と呼び出されます。まず write メソッドによって、変数 line2 に代入された文字列をファイルに書き出すことが指定されます。続いて flush メソッドによって、変数 bwriter に保持されていた文字列が実際にファイルに書きだされ、そして newLine メソッドによって改行が指定されます。

さて、このプログラムを実行した後に、テキストエディタで bmi.txt の中身を表示してみてください。ファイルの中身は、以下のようになっていますでしょうか？

```
w=57.0 h=1.64 BMI=21.192742415229034
w=59.0 h=1.66 BMI=21.410944984758313
w=64.0 h=1.77 BMI=20.428357113217785
w=70.0 h=1.73 BMI=23.38868655818771
w=79.0 h=1.81 BMI=24.114038033027075
w=41.0 h=1.49 BMI=18.467636592946263
w=45.0 h=1.55 BMI=18.730489073881373
w=47.0 h=1.59 BMI=18.59103674696412
w=62.0 h=1.59 BMI=24.524346347059055
w=58.0 h=1.67 BMI=20.796729893506402
```

# 9　オブジェクト指向に関連する用語群

本章では、Java 言語が踏襲している「オブジェクト指向」という考え方と、それに関連する典型的な用語をいくつか紹介します。

## 9.1　オブジェクト指向

**オブジェクト指向** とは、プログラムが処理するデータ（数値）や処理手順を、現実世界の「モノ」になぞらえた考え方のひとつです。

例えば 1.1 以降、私たちは何度か、BMI を算出するプログラムを使用してきました。いま、このプログラムを「BMI 電卓」という「モノ」にたとえてみましょう。皆さんはこの電卓を実現するときに、どんな部品が必要になりますか？おそらく多くの人は、以下の 3 種類の部品が必要だと考えるでしょう。

- 体重や身長を入力するためのボタン部品。
- BMI を計算するための計算部品。
- BMI を計算した結果を出力するための表示部品。

このように、「モノ」はいくつかの部品で構成されます。いままで習得してきた Java 言語では、ちょうど「モノ」がクラスに相当し、「部品」がメソッドや変数に相当する、と考えることができます。

では、複数の「モノ」を組み合わせて使うことを考えてみましょう。例えば折れ線グラフを描く装置と BMI 電卓を組み合わせると、長期間にわたる BMI 値の変動を描くことができるでしょう。それと同様に、Java 言語では、多くのクラスを組み合わせることで、さらに高度な処理を実現できます。

このように、プログラムによる処理を、「モノの組み合わせ」として考えることが、「オブジェクト指向」の基本的な考え方となります。

皆さんは Java 言語以外の言語で、**フローチャート**などの形で処理の手順を描く機会があるかもしれません。オブジェクト指向は、処理手順に基づいて組み立てる旧来のプログラム言語に対比する新しい考え方として成長した、という歴史的経緯のある考え方です。

### カプセル化

上述のとおり、本章ではオブジェクト指向を BMI 電卓に比喩して説明しています。ここで BMI 電卓という「モノ」を、「目に見えるモノ」と「目に見えないモノ」に分類して考えてみましょう。あくまでも一例として、以下のように分類できるのではないでしょうか。

**目に見えるモノ：** 身長や体重の入力部品、BMI 算出結果の出力部品。

**目に見えないモノ：** BMI 計算式、BMI 計算途中の値。

こうしてみると、モノの中と外での情報のやり取りは目に見えるように、そしてモノの内部処理は目に見えないように設計するのが一般的であろう、と考えることができます。

Java 言語でもこれと同様に、クラスの中と外で情報をやり取りする部分は積極的に他のクラスやパッケージに公開し、クラスの中で閉じた形で用いる部分は他のクラスやパッケージに公開しない、という考え方が普及しています。このような考え方を **カプセル化** と呼びます。

7.4 節にて public と private というキーワードを説明しましたが、これらのキーワードはまさに、カプセル化を実現するために、積極的に公開するメソッドや変数と、公開しないメソッドや変数を指定するためのキーワード、といえるかと思います。

## 9.2 派生と継承：extends

Java 言語では、クラスに「親子」という概念を持ち込むことができます。そして、ある「親クラス」の性質（具体的にはメソッドや変数）を「子クラス」にも持たせることができます。このことを「親クラスから子クラスへの**継承**」といいます。また、このときの子クラスを「親クラスから**派生**したクラス」といいます。

派生と継承を実現するために、Java 言語では extends というキーワードを用意しています。

一例として、以下のように Parent という親クラスを定義したとします。

```
public class Parent {
   public void calcSquare(double a) {
     double square = a * a;
     System.out.println("Square=" + square);
   }
}
```

続いて一例として、以下のように Child という子クラスを定義します。

```
public class Child extends Parent {
   public void calcCube(double a) {
     double cube = a * a * a;
     System.out.println("Cube=" + cube);
   }
}
```

ここで public class Child の次にある「extends Parent」に注目してください。これは、Child クラスは「Parent クラスの派生クラス」であり、「Parent クラスの性質を継承するクラス」であり「Parent クラスを拡張するクラス」であることを示しています。具体的には、Child クラスにも Parent クラスのメソッドである calcSquare が継承されて使えるようになります。

この Child クラスを呼び出す main メソッドの例を、以下に示します。

```
public class UseChild {
   public static void main(String[] args) {
     double a = 3.0;
     Child child = new Child();
     child.calcSquare(a);
     child.calcCube(a);
   }
}
```

この main メソッドでは、Child クラスの変数 child のメソッド calcSquare および calcCube を呼び出しています。calcSquare は元々Parent クラスのメソッドですが、Child クラスがこれを継承しているので、あたかも Child クラスのメソッドであるかのように使うことができます。

例えばBMI電卓の新商品が、旧商品の機能を受け継いだまま、新しい機能を追加搭載したとしましょう。このとき、旧商品を親、新商品を子と考えると、旧商品の機能は親から子に受け継がれた上で、新商品には新機能が搭載される、と考えられます。Java言語における派生と継承の考え方は、このような形で現実世界の「モノ」にも共通する考え方である、といえます。

## 9.3 オーバーライド：final と abstract

Java言語では逆に、親クラスから派生した子クラスにて、あえて親クラスのメソッドと同名のメソッドを新しく定義しなおすことができます。このようにして新しく定義することを、**オーバーライド** といいます。

一例として、以下のように Parent という親クラスを定義したとします。

```
public class Parent {
   public void print() {
     System.out.println("I am the Parent.");
   }
}
```

続いて一例として、以下のように Child という子クラスを定義します。

```
public class Child extends Parent {
   public void print() {
     System.out.println("I am the Child");
   }
}
```

Childクラスは Parentクラスの派生クラスなので、本来でしたら Parentクラスの printメソッドを継承するはずです。しかし、この Childクラスでは、printメソッドをオーバーライドしているため、Parentクラスの printメソッドは実行されずに、Childクラスの printメソッドが実行されます。

このオーバーライドという機能は、一例として以下のような状況にて有効に活用できます。仮に Parentクラスに10個以上のメソッドがあったとします。これの派生クラスである Childクラスでも、Parentクラスの大半のメソッドを継承したままで使いたいのですが、どうしても1,2個だけ、メソッドの中身を書き換えたいとします。このとき、Parentクラスとの継承関係を絶つよりも、その1,2個のメソッドだけを書き換えたほうが、プログラム開発において格段に合理的です。このような状況において、オーバーライドは有効に活用できるといえます。

### final

親クラスのメソッドに **final** と定義することで、子クラスによるオーバーライドを禁止することができます。

一例として、以下のように Parent という親クラスを定義したとします。

```
public class Parent {
   public final void print() {
     System.out.println("I am the Parent.");
```

```
    }
}
```

このとき print メソッドには final が指定されているので、Parent クラスを継承する派生クラスにて print メソッドを再定義してしまうと、コンパイル（javac 命令の実行）にてエラーを生じてしまいます。

**abstract**

親クラスのメソッドに **abstract** と定義することで、子クラスによるオーバーライドを強制することができます。

一例として、以下のように Parent という親クラスを定義したとします。

```
public abstract class Parent {
   public abstract void print() {
   }
}
```

このとき print メソッドには abstract が指定されているので、Parent クラスを継承する派生クラスにて print メソッドが再定義されていないと、コンパイルにてエラーを生じてしまいます。

なお、abstract が指定されたメソッドを含むクラスは、クラス自体にも abstract 指定をしないといけません。上述の例では、abstract 指定された print というメソッドを含む Parent クラスは、クラス自体にも abstract 指定しないといけないので、public abstract class Parent というように記述されています。

このように、abstract 指定されたクラスやメソッドを、**抽象クラス** あるいは**抽象メソッド** と呼びます。

Java 言語では、抽象クラスや抽象メソッドなどを用いて、とりあえず抽象的にクラス名やメソッド名を定義しておき、メソッドの具体的な中身は継承クラスにまかせる、というような技法がよく用いられます。これは例えば、プログラムを 2 人以上の人で開発するときに、最初の担当者がとりあえず抽象クラスを定義しておき、次の担当者が継承クラスを開発する、というような分担作業をするときに便利です。

## 9.4 仕様と実装：interface と implements

Java 言語では、前節で紹介した抽象クラスとは別に、**インタフェース** という定義に基づいて、抽象的にクラスやメソッドを定義することができます。

ここで早速ですが、インタフェースの例を示します。

```
public interface Parent {
   public void print();
   public void calcSquare(double a);
}
```

いままでクラスを定義するときには、1 行目はお決まりのように public class というような単語が並んでいましたが、インタフェースを定義するときには class という単語の代わりに **interface** を使います。

そしてインタフェースでは、メソッドの名前、引数、返り値は定義するだけで、その中身（中カッコで括られた処理本体）は一切記述しません。

では、その中身はどのように定義すればいいのでしょうか。以下にその例を示します。

```
public class MyParent implements Parent {
   public void print() {
     System.out.println("I calculate Square.");
   }
   public void calcSquare(double a) {
     double square = a * a;
     System.out.println("Square=" + square);
   }
}
```

このプログラムの 1 行目に着目してください。1 行目の前半 `public class MyParent` は、今まで示した多くのプログラムと同様に、このクラスが `MyParent` というクラスであることを示しています。続いて 1 行目の後半 `implements Parent` は、`Parent` というインタフェースの中身を定義していることを示します。このように、インタフェースで定義されたメソッドの中身を記述するクラスには、**implements** という単語を用いて、「どのインタフェースの中身を記述したか」を指定します。

このようなインタフェースとクラスの対比的な関係を、しばしば**仕様** と**実装** という単語で表現することがあります。「仕様」とは多くの場合、規格や規則を定めたものを指します。Java 言語の場合、メソッドの名前、引数、返り値などの規格が仕様に相当します。そして、その処理本体が実装に相当します。

Java 言語に限らず、多くのプログラム言語の開発では、まず仕様が策定され、続いてプログラマは実装を開発します。これは BMI 電卓にたとえると、まずデザインや機能が決められ、続いてその機能を実現するための中身（電子回路など）を開発する、という手順に似ています。

Java 言語が採用している `interface` および `implements` の機能は、現実世界にも浸透している「仕様と実装」という考え方を実現しやすくした、といえます。

また、インタフェースは抽象クラスと同様に、複数の人でプログラムを開発するときに、非常に有用です。例えば最初の開発者が、とりあえずインタフェースを定義しておき、次の担当者がその中身を記述するクラスを開発する、というような分担作業をするときに便利です。インタフェースと抽象クラスは以上の意味で役割が類似していますが、強いて違いをあげるとしたら、インタフェースは仕様だけを定義する仕組みであるのに対して、仕様だけでなく実装の一部を継承したり再定義したりする、というようなことが必要な場合に抽象クラスを活用します。

## 9.5 【サンプルプログラム】複数の相性占い師

それでは、「仕様と実装」の考え方を端的に表したプログラムの例を示します。6.4 節で紹介した「相性占い」のプログラムを大きく拡張したものです。かなりプログラムが大きくなりますが、頑張ってください。

まず、下のインタフェースを、`Uranaishi.java` というファイルで保存してください。このインタフェースは、2 人の月日を代入して、相性を返すメソッドを定義します。

皆さんがプログラムの発注者になって、いろんな人に相性計算のプログラムを書いてほしい、という状況を想像してください。このとき、

**下のインタフェースをプログラマに公開するだけで、プログラマたちは相性計算のメソッドの名前、引数、返り値を知ることができます。** これによって、いろんなプログラマが、統一された形式で相性計算の

プログラムを書くことができます。

```
public interface Uranaishi {

   // 2 人の月日を代入して、相性を返すメソッドを定義する
   public int calculateChemistry(int month1, int day1, int month2, int day2);
}
```

続いて、上のインタフェースにしたがって、2 人のプログラマが相性計算のプログラムを開発した、という状況を想像してください。

まず Alice さんが、インタフェース Uranaishi の記述にしたがって、UranaishiAlice というクラスを開発したとします。このプログラムは「implements Uranaishi」と書いてあることからもわかるように、インタフェース Uranaishi の記述にしたがって相性計算のプログラムを書いています。

では、下のプログラムを、UranaishiAlice.java というファイルに保存してください。

```
public class UranaishiAlice implements Uranaishi {

   // 元旦からの合計日数を求める
   int calculateTotalDays(int month1, int day1) {
     int total = 0;
     int days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

     // 誕生月の前の月までの合計日数を求める
     for(int i = 0; i < (month1 - 1); i++) {
       total += days[i];
     }

     // 合計日数に日を足す
     total += day1;

     // 合計日数を返す
     return total;
   }

   // 2 人の月日を代入して、0〜100 パーセントで相性を求める
   public int calculateChemistry(int month1, int day1, int month2, int day2) {
     int chemistry = 0;

     // 人物 A の誕生日の元旦からの合計日数を求める
     int total1 = calculateTotalDays(month1, day1);
     // 人物 B の誕生日の元旦からの合計日数を求める
     int total2 = calculateTotalDays(month2, day2);

     // 相性の算出式 （当てずっぽうな算出式なので信頼できるものではない）
     chemistry = (total1 + total2) % 101;
```

```
    // 相性を返す
    return chemistry;
  }
}
```

続いて同様に Bob さんが、インタフェース Uranaishi の記述にしたがって、UranaishiBob というクラスを開発したとします。

では、下のプログラムを、UranaishiBob.java というファイルに保存してください。

```
public class UranaishiBob implements Uranaishi {

  // 2 人の月日を代入して、0〜100 パーセントで相性を求める
  public int calculateChemistry(int month1, int day1, int month2, int day2) {
    int chemistry = 0;

    // 人物 A の月と日を足す
    int total1 = month1 + day1;
    // 人物 B の月と日を足す
    int total2 = month2 + day2;

    // 相性の算出式 （当てずっぽうな算出式なので信頼できるものではない）
    chemistry = (total1 + total2) * 100000 % 101;

    // 相性を返す
    return chemistry;
  }
}
```

そして最後に、かつて作成した Uranai.java を UranaiMulti.java にコピーして、以下のように書き換えてください。

```
import java.io.*;

public class UranaiMulti {

  // メインメソッド
  public static void main(String[] args) {

    // 2 人の誕生日の月と日を代入する変数
    int month1 = 0, month2 = 0, day1 = 0, day2 = 0;

    // 相性の算出結果を代入する変数
    int chemistry = 0;
```

```
      // 人物 A の月、日、人物 B の月、日、の順に標準入力する
      try {
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);
        System.out.println("Month of birthday A");
        month1 = Integer.parseInt(br.readLine());
        System.out.println("Day of birthday A");
        day1 = Integer.parseInt(br.readLine());
        System.out.println("Month of birthday B");
        month2 = Integer.parseInt(br.readLine());
        System.out.println("Day of birthday B");
        day2 = Integer.parseInt(br.readLine());
        br.close();
      }
      catch (Exception e) {
        System.out.println(e);
      }

      // Alice による相性算出結果を表示する
      UranaishiAlice u1 = new UranaishiAlice();
      chemistry = u1.calculateChemistry(month1, day1, month2, day2);
      System.out.println("By Alice:  chemistry= " + chemistry);

      // Bob による相性算出結果を表示する
      UranaishiBob u2 = new UranaishiBob();
      chemistry = u2.calculateChemistry(month1, day1, month2, day2);
      System.out.println("By Bob:  chemistry= " + chemistry);

   }
}
```

これを実行すると、例えばこんな感じで、Alice と Bob による 2 人の相性計算結果が出力されます。

```
Month of birthday A
1
Day of birthday A
3
Month of birthday B
4
Day of birthday B
12
By Alice:  chemistry= 4
By Bob:  chemistry= 99
```

このプログラミングのミソは、Alice と Bob が 2 人とも、インタフェース Uranaishi にしたがって相性計算のプログラムを書いていることです。これによって、UranaiMulti クラスでは、Alice と Bob の相性計算を、そっくりな形で呼び出すことができるので、プログラミングが簡単になります。

ここまでできた人は、近くの友人たちで集まって、以下の課題も挑戦してみてください。

1. 自分だけの相性計算のクラスを作ってみてください。例えば Takayuki という名前の人が作成する相性計算のクラスを、UranaishiTakayuki という名前のクラスにしてみてください。Alice とも Bob とも、また周囲の友人とも違う、自分だけの計算式を考えてプログラミングするのが重要です。
2. 友人どうしで、メールで送り合うか、あるいは USB メモリで交換するなどして、相性計算のクラスを共有してください。
3. UranaiMulti クラスに、Alice や Bob だけでなく、いろんな人による相性計算のクラスをたくさん追加して、いろんな相性計算結果を出してください。

Java 言語の面白い点の一つとして、インタフェースさえ公開すれば、いろんな人が互いに利用し合えるプログラムを開発しやすくなる、という点があります。例えば相性計算であれば、いろんな人による相性計算のプログラムを集めて、信用できる相性計算だけを採用する、時と場合によって相性計算を入れ替える、などといった多彩な楽しみ方が考えられるかと思います。

# 10　アルゴリズムを考えながらプログラムを書こう：ポーカーゲームを例にして

前章までで、非常に多くのプログラムを開発してきました。しかし本講義での前章までのプログラミングは、主に文法を覚えるためのものでした。英語の試験に文法問題と長文問題があるように、プログラミングは文法を覚えるだけでなく長文を書けるようになる必要があります。むしろ、プログラミングの現場には、文法の試験などありません。皆さんは研究や仕事でプログラミングの作業に就くことになったら、ひたすら長文を書くことが必要となります。

そこで本章では、自分で長文を考えるように、あえて不親切に、あまりプログラムの詳細を見せずに、処理手順（アルゴリズム）を考えながらプログラムを考えてもらう練習をします。本章では練習の例題として、トランプのポーカーを実現するプログラムを、数段階に分けて開発します。

## 10.1　ポーカーとは

ポーカーとは、トランプの中から 5 枚を引き、その数字とマークから得られる「ハンド」という組み合わせの強さを競うものです。ハンドには弱い順に、以下のようなものがあります[9]。

**ワンペア** 同一数字のカード 2 枚のペア一組（残り 3 枚は何でもよい）。

**ツーペア** 同一数字のカード 2 枚からなるペアが二つ（残り 1 枚は何でもよい）。

**スリーカード** 同一数字のカード 3 枚（残り 2 枚は何でもよい）。

**ストレート** 5 枚のカードの数字が連続している。

**フラッシュ** 5 枚全てが同じマーク（数字は何でもよい）。

**フルハウス** ワンペアとスリーカードの組み合わせ。

**フォーカード** 同一数字のカード 4 枚（残り 1 枚は何でもよい）。

**ストレートフラッシュ** 5 枚のカードが連番で、なおかつ全て同じマーク。

本科目では、計算機が自動的に 5 枚のカードを引き、そのハンドを判定して表示する、というプログラムをつくることにします。

## 10.2　プログラミングその 1：5 枚のカードを引く

まずテキストエディタで `Poker1.java` というファイルを作ってください。そして `Poker1.java` に以下のようなプログラムを書いてください。

```
// ポーカーゲーム
// その 1： 5 枚のカードを、重複がないように引く
public class Poker1 {

  // 5 枚のカードの番号を表示する
  public static void print(int[] mark, int[] number) {
    (中略)
```

[9] もっと強いハンドに「ロイヤルストレートフラッシュ」がありますが、ここでは省略します。

```
  }

  // 5 枚のカードを引き、マークと番号を決定する
  public static void pick(int[] mark, int[] number) {
    (中略)
    // 結果を表示する
    print(mark, number);
  }

  // main メソッド
  public static void main(String[] args) {
    int[] mark = new int[5];
    int[] number = new int[5];

    // 5 枚のカードを引く
    pick(mark, number);
  }
}
```

ここで、main メソッドの変数 mark には、5 枚のカードのマーク（スペード、クローバー、ハート、ダイヤの 4 種類）のいずれかを、1 から 4 までの整数におきかえて記録することにします。また、変数 number には、5 枚のカードの番号（1 から 13 までの整数）を記録することにします。

pick メソッドの中の（中略）と書いた部分では、3.1 節に出てきた戦争ゲームと同じ要領で、5 枚のカードの各々について、乱数を使ってマークと番号を決定するプログラムを書いてください。

また、print メソッドの中の（中略）と書いた部分では、5 枚のカードのマークと番号を画面に表示するプログラムを書いてください。ただし、スペードを表す 1 は「S」に、クローバーを表す 2 は「C」に、ハートを表す 3 は「H」に、ダイヤを表す 4 は「D」に置き換えて表示してください。

私のプログラムでは、これを実行すると、以下のように表示されます。

```
S 9
S 13
D 8
S 2
H 2
```

これで、「5 枚のカードを引いた」のと同じ効果が出たことになりますが、しかし重要なことに気をつけないといけません。皆さんの現段階のプログラムでは、ひょっとしたら、マークも番号も全く同じカードを 2 枚引いてしまうかもしれません。現実のポーカーでは、それはありえないので、同じカードを 2 枚引かないような判定文をプログラムに加える必要があります。以下に、その処理のヒントを書きます。

$i$ 番目のカードのマークと番号が決まりましたら、$j$ 番目（$0 \leq j \leq (i-1)$）のカードについて、マークと番号を照合してみてください。もし $i$ 番目と $j$ 番目のカードが、マークも番号も両方とも同一だったら、$i$ 番目のカードのマークと番号を、乱数を使ってもう一度決定してください。

ここまで書き終えたら、Poker1.java の動作を再確認してください。

## 10.3　プログラミングその 2：5 枚のカードを小さい順に並べ替える

ここでは、ポーカーのハンドを判定するために、5 枚のカードを小さい順に並べ替える処理を書き加えます。この並べ替えの処理を **ソート**ともいいます。ソートは非常に多くのプログラムで頻繁に用いられます。

まず Poker2.java を、以下のように書いてください。そして新しく加わった sotr というメソッドの中に、並べ替えのプログラムを書いてください。

```
// ポーカーゲーム
// その 1： 5 枚のカードを、重複がないように引く
public class Poker2 extends Poker1 {

   // 小さい順に並べ替える
   public static void sort(int[] mark, int[] number) {
     (中略)
     // 結果を表示する
     print(mark, number);
   }

   // main メソッド
   public static void main(String[] args) {
    (中略)
     // 5 枚のカードを引く
     pick(mark, number);
     // 小さい順に並べ替える
     sort(mark, number);
   }
}
```

並べ替えの処理手順は以下の通りです。まず $i$ 番目のカードの番号を見てください。そして $j$ 番目のカードについて、$j = (i+1)$ から $j = 4$ 番目まで順番に、$i$ 番目のカードとの大小を比較してください。もし $i$ 番目のカードより $j$ 番目のカードの番号のほうが小さかったら、$i$ 番目と $j$ 番目のカードを **マーク・番号の両方とも**　入れ替えてください。

マークや番号を入れ替えるプログラムをつくるには、ちょっとした工夫が必要です。ヒントとして、mark と number の他に、もう 1 個変数を用意する必要があります。これを考えてみてください。

$i$ に 0,1,2,3,4 の値をそれぞれ順に代入して、同様な処理を反復すれば、カードは小さい順に並び替えられるはずです。

以上の処理を、sort メソッドの（中略）と書かれた部分に挿入して、動作確認してみてください。私のプログラムでは、これを実行すると、以下のように表示されます。前半が並び替え前の 5 枚、後半がそれを並べ替えた結果です。

```
D 5
H 11
D 10
C 2
D 1
```

```
D 1
C 2
D 5
D 10
H 11
```

## 10.4 プログラミングその 3：ハンドを判定する (1)

ここでは、ワンペア、ツーペア、スリーカード、ストレート、の各々のハンドが該当するかを判定するメソッドを、Poker3.java の中につくりましょう。

一例としてワンペアが該当するかを判定するメソッドは、以下のように作ってください。

```
  // ワンペアを判定する
 public static boolean onepair(int[] mark, int[] number) {
   if(...)  {
     System.out.println("...  OnePair!");
     return true;
   }
   return false;
 }
```

このメソッドでは、if 文にてワンペアと判定されるようであれば、System.out.println 文を用いてワンペアであることを表示し、return true; の行で true を返します。if 文に入らない場合（ワンペアでない場合）は、return false; の行で false を返します。

ツーペア、スリーペア、ストレートについても、同様にメソッドを作ってください。

ここで、前節で開発した処理により、カードは番号の小さい順に並び替えられています。気がついたかと思いますが、この性質を利用することで、ワンペア、ツーペア、スリーカード、ストレート、いずれも判定のプログラムを簡単に記述できるはずです。

これらができましたら、Poker3.java を以下のように書いてください。

```
// ポーカーゲーム
// その 3： 5 枚のカードを、重複がないように引き、小さい順に並べ替え、いくつかのハンドを判定する
public class Poker3 extends Poker2 {

   // ストレートを判定する
   public static boolean straight(int[] mark, int[] number) {
     (中略)
   }

   // スリーカードを判定する
   public static boolean threecard(int[] mark, int[] number) {
     (中略)
```

```
  }

  // ツーペアを判定する
  public static boolean twopair(int[] mark, int[] number) {
    (中略)
  }

  // ワンペアを判定する
  public static boolean onepair(int[] mark, int[] number) {
    (中略)
  }

  // ハンドを判定する
  public static void judge(int[] mark, int[] number) {
    boolean ret;

    ret = straight(mark, number);
    if(ret == false) {
      ret = threecard(mark, number);
    }
    if(ret == false) {
      ret = twopair(mark, number);
    }
    if(ret == false) {
      ret = onepair(mark, number);
    }
  }

  // main メソッド
  public static void main(String[] args) {
    (中略)
    pick(mark, number);
    sort(mark, number);
    judge(mark, number);
  }
}
```

この中の judge メソッドでは、まずストレートを判定します。ストレートに該当しなければ、スリーカードを判定します。スリーカードにも該当しなければ、ツーペア、ワンペア、の順に判定します。このようにポーカーでは、高いハンドから順に判定することになります。

私のプログラムでは、これを実行すると、以下のように表示されます。

```
D 1
S 9
C 6
```

```
H 1
H 11

D 1
H 1
C 6
S 9
H 11

...  OnePair!
```

ここまで開発できましたら、何度も何度も、繰り返し実行してみてください。そして、ハンドが毎回必ず合っているか、よく見てよく確認してください。

## 10.5 プログラミングその 4：ハンドを判定する (2)

今度は残りのハンドとして、フラッシュ、フルハウス、フォーカード、ストレートフラッシュ、の 4 種類を、Poker4.java に書いてください。前節と同じように、各々のハンドに該当するなら return true; さもなければ return false; として、true または false のいずれかを返します。

Poker4.java には、上述の新しい 4 種類のハンドの他に、judge メソッドと main メソッドをつくってください。main メソッドは Poker3.java と全く同一です。judge メソッドは Poker3.java と似たような作り方で、合計 8 種類のハンドについて判定をしてください。判定する順番は、ストレートフラッシュ、フォーカード、フルハウス、フラッシュ、ストレート、スリーカード、ツーペア、ワンペアの順です。

## 10.6 プログラミングその 5：各々のハンドが何回ずつ出現するか集計する

最後に、「5 枚のカードを引いて、並べ替えて、ハンドを判定して」という処理を何度も繰り返して、各々のハンドが何回ずつ出現するか集計する、というプログラムをつくりましょう。以下の通り、Poker5.java を作ってください。

```
public class Poker5 extends Poker4 {
   public static int count0 = 0, count1 = 0, count2 = 0, count3 = 0,
            count4 = 0, count5 = 0, count6 = 0, count7 = 0;

   // 役を判定する
   public static void judge(int[] mark, int[] number) {
     (中略)
   }

   // main メソッド
   public static void main(String[] args) {
     (中略)
   }
```

}

このプログラムでは、メソッドの内部に入っていない変数 count0〜count7 を、各々のハンドの出現回数を集計するために使ってください。この出現回数の集計は、judge メソッドの中で行ってください。具体的には、ある特定のハンドに該当するときには、その該当するハンドに対応する変数（count0〜count7 のいずれか）に 1 を加えてください。

この count0〜count7 のように、メソッドの外側に変数を出すことで、メソッドを呼び出して返ってきてからも、変数に代入された値を保存し続けることができるようになります。このプログラムの場合、judge メソッドを何度も呼び出して、何度も返ってきても、集計値は保存し続ける必要があるために、count0〜count7 の各変数をメソッドの外に出すことになります。

main メソッドでは、pick, sort, judge の各メソッドを、何度も繰り返し呼んでください。そして、その反復が終了したら、各々のハンドの出現回数を表示してください。なお、反復回数が非常に多い場合には、画面表示に大きな時間がかかってしまいます。この場合に限り、Poker1.java の print メソッドの中にある System.out.println 文の行頭に//をつけるなどして無効化（コメントアウト）してください。

私のプログラムでは、これを実行すると、以下のように集計結果が表示されます。ちなみに私のプログラムでは、1 万回反復しています。

```
### StraightFlash:  1
### FourCard :  1
### FullHouese :  15
### Flash :  16
### Straight :  31
### ThreeCard :  222
### TwoPair :  484
### OnePair :  4273
```

このプログラムを何度か繰り返してみればわかりますが、上位のハンドになればなるほど、出現回数が少なくなります。ポーカーのハンドが、よくできたものであることが、これでわかるかと思います。

# 11 【課題】ここまでに紹介されたプログラムを拡張してみよう

冒頭でも書きましたとおり本書籍は、筆者が非常勤講師としてプログラミングの演習科目を担当した際の講義資料をそのまま書籍化したものです。その演習科目は理工学系学部の 1 年生前期の科目であり、大半の学生にとってプログラミングは全くの初めての経験でした。そのような学生の最初の提出課題はどのようなものだったのか、本章にそのまま掲載いたします。Java 言語を学習中の皆さまの参考になれば幸いです。

## 11.1 課題 1: おみくじの拡張

2.1 節で説明したおみくじを、以下のとおり書き換えて提出して下さい。

- クラス名を Omikuji2 にする。つまり提出するプログラムファイルは Omikuji2.java である。
- おみくじを 100 回反復し、1 回ずつ回答を出す。
- 「大吉」「中吉」「小吉」「凶」がそれぞれ何回ずつ出たか集計し、プログラム終了前に各々の回数を表示する。

以下に、教員が書いたプログラムの実行例をお見せします。このプログラムでは、「大吉」「中吉」「小吉」「凶」のいずれかの結果を 1 回ずつ表示し、最後に各々の出現回数を表示しています。

```
Kyo
Daikichi
Daikichi
Kyo
(略)
Daikichi:24 Chukichi:24 Shokichi:28 Kyo:24
```

## 11.2 課題 2: 戦争ゲームの復習

3.1 節で説明した戦争ゲームを、以下のとおり書き換えて提出してください。

- クラス名を Senso2 にする。つまり提出するプログラムファイルは Senso2.java である。
- カードの枚数ではなく、点数を競うゲームにする。
- 以下の説明では、人物 A が出したカードの数字を $a$，人物 B が出したカードの数字を $b$ とする。
- $a > b$ であれば、人物 A は点数 $(a - b)$ を獲得する。
- $a < b$ であれば、人物 B は点数 $(b - a)$ を獲得する。
- $a == b$ であれば、その次の回に負けた人の点数は半分（端数切捨て）になる。
- 一方の点数が他方の点数に 100 点以上の差をつければゲーム終了。
- ただし 100 回繰り返しても決着がつかなければ、そこでゲーム終了。
- ゲームの内容の変化にあわせて、System.out.println() 行のカッコの中も記述を変えること。
- System.out.println() 行にて、いま何回目のゲームであるかを明記すること。

- ゲーム終了の時点で System.out.println() 行を追加して、人物 A と人物 B のどちらが点数が多かったか、を明記する表示を加えること。

以下に、教員が書いたプログラムの実行例をお見せします。このプログラムでは、まず何回目であるかを冒頭に表示して、続いて毎回のカードの数字を a=および b=の後に表示して、その結果としての A および B の点数を A's score=および B's score= の後に表示しています。以下の例では、52 回目で $a == b$ となり、53 回目で A が負けて、A の点数が 0 になっています。しかしその後、84 回目で $a == b$ となり、85 回目で B が負けて B の点数が 0 になっています。そして 86 回目で 100 点差がついて、GAME OVER ... という表示が出て、最後に「A が勝った」という表示が出ます。

```
1:  a=5 b=13 A's score=0 B's score=8
2:  a=2 b=5 A's score=0 B's score=11
3:  a=4 b=6 A's score=0 B's score=13
（略）
51:  a=11 b=6 A's score=43 B's score=60
52:  a=b=6 A's score=43 B's score=60
53:  a=5 b=7 A's score=21 B's score=62
（略）
83:  a=2 b=5 A's score=91 B's score=53
84:  a=b=10 A's score=91 B's score=53
85:  a=10 b=7 A's score=94 B's score=26
86:  a=7 b=1 A's score=100 B's score=26
GAME OVER ...  A's score=100 B'score=26
...  A wins
```

## 11.3 課題 3: 数当てゲーム

以下の要件を満たすプログラムを開発して提出して下さい。

- クラス名を Kazuate にする。つまり提出するプログラムファイルは Kazuate.java である。
- プログラムは最初に、1 から 200 までの間の任意の整数を 1 個作る。作り方は 3.1 節を参照。この整数は画面表示しないこと。以後、この値を「正解」と呼ぶ。
- プログラムは続いて、1 から 200 までの任意の整数をもう 1 個作り、「正解」と比べてその値が大きいか小さいかを、ヒントとして表示する。
- つづいて、この正解がいくつかであるか、キーボード入力させて当てさせる。正解を当てられるまで、何度でも反復させる。キーボード入力方法は 6.2 節および 6.4 節参照。
- 当たらなかったときは、キーボード入力した値と正解を比べて、どちらが大きいかを画面出力する。
- 当たったら、それが正解であることを告げるとともに、正解を当てるまで何回かかったかを表示して、反復処理から抜け出す。反復処理の抜け出し方は、3.6 節参照。

以下に、教員が書いたプログラムの実行例をお見せします。このプログラムでは最初に、正解は 102 より小さい、と表示します。Please input the number が表示されたらキーボード入力し、その値と比べて

正解が大きいか小さいかを表示する、という処理を反復しています。正解を当てられれば、「BINGO!!」と表示し、そして何回かかったかを表示します。

```
Answer is smaller than 102
Please input the number
50
...  answer is smaller than 50
Please input the number
25
...  answer is larger than 25
Please input the number
37
...  answer is smaller than 37
Please input the number
32
...  answer is smaller than 32
Please input the number
28
...  answer is larger than 28
Please input the number
30
...  BINGO!!  answer is 30
...  You took 6 times.
```

## 11.4 課題 4: 相性占いの拡張版

6.4 節で説明した相性占いのプログラムを、以下のとおり書き換えて提出してください。なお全面的に、8.1 節および 8.2 節を参考にしてください。またこの問題では、あくまでも便宜上、「男性と女性の相性を計算する」という問題であることにします。

- クラス名を Uranai2 にする。つまり提出するプログラムファイルは Uranai2.java である。
- 相性を占いたい男性の月、日、女性の月、日、の 4 つの数字を 1 行にまとめたファイルを作り、birthday.txt というファイル名で保存する。ファイルの作成方法については 8.1 節を参照。
- このファイルを入力し、ファイル中の各行について相性を算出する。そして相性の算出結果を、aishou.txt という名前の別のファイルに出力する。aishou.txt の各行には、男性の月、日、女性の月、日、相性、の順に 5 つの数字を記載し、最後に相性値が 50 以上であれば Good、50 未満であれば Bad と記載する。ファイルの入力については 8.1 節を参照。ファイルの出力については 8.2 節を参照。

以下に、教員が書いたプログラムの実行例をお見せします。ここでは仮に、birthday.txt には以下のように男女 8 組の誕生日が書かれているとします。

```
1 3 4 12
5 8 10 11
5 31 2 20
```

```
7 16 12 1
8 21 1 19
9 6 7 8
10 25 9 29
12 11 5 30
```

これに対して、aishou.txt には以下の通り、各行に記載された 8 組の男女の相性を出力しています。

```
Month=1 Day=3 Month=4 Day=12 AISHOU=4 Bad!
Month=5 Day=8 Month=10 Day=11 AISHOU=8 Bad!
Month=5 Day=31 Month=2 Day=20 AISHOU=0 Bad!
Month=7 Day=16 Month=12 Day=1 AISHOU=27 Bad!
Month=8 Day=21 Month=1 Day=19 AISHOU=50 Good!
Month=9 Day=6 Month=7 Day=8 AISHOU=34 Bad!
Month=10 Day=25 Month=9 Day=29 AISHOU=65 Good!
Month=12 Day=11 Month=5 Day=30 AISHOU=91 Good!
```

### 上級者向けの課題

余裕のある人は、以下の課題も解いてみましょう。

- クラス名を Uranai3 にする。
- 各々の男性について、全ての女性との相性を計算し、最も相性の高いものを選んで表示する。

以下に、教員が書いたプログラムの実行例による、aishou.txt ファイルの出力結果の一部を示します。この出力結果では、例えば 1 行目の場合、dansei=0 が「0 番目の男性」を表し、josei=5 が「5 番目の女性」を表し、aishou=91 が「相性 91 パーセント」を表しています。つまり、0 番目の男性について、8 人の女性全員との相性を計算したところ、5 番目の女性が最も相性が高く、その数字は 91 パーセントだった、ということを意味します。

```
dansei=0 josei=5 aishou=91
dansei=1 josei=6 aishou=97
dansei=2 josei=7 aishou=99
dansei=3 josei=0 aishou=97
dansei=4 josei=2 aishou=82
dansei=5 josei=2 aishou=98
dansei=6 josei=0 aishou=97
dansei=7 josei=2 aishou=93
```

### もう 1 つ、上級者向けの課題

余裕のある人は、以下の課題も解いてみましょう。

- クラス名を Uranai4 にする。

- ある男性の誕生日をプログラム中に直接書き込んで、1 月 1 日から 12 月 31 日までの全ての誕生日について女性との相性を計算し、相性が 100%になる誕生日を表示する。
- 男女の誕生日は birthday.txt に記載しないこと。つまり birthday.txt というファイルを読まないこと。

以下に、教員が書いたプログラムの実行例による、aishou.txt ファイルの出力結果の一部を示します。プログラム中に書き込んだ男性の誕生日を 1 月 3 日とした場合に、以下の month, day で誕生日を記載された 4 月 7 日生まれ、7 月 17 日生まれ、10 月 26 日生まれの女性との相性が 100%になっていることを表しています。

```
month=4 day=7
month=7 day=17
month=10 day=26
```

# 12 【付録 A】プログラミングとは

本章はプログラミングの概念を解説するものであり、筆者の非常勤先の講義資料では第 1 章として掲載し、第 1 週に講義していたものです。しかし既に内容的にもあまり新しくないこと、プログラミング実習に専念した資料として本書を再編集していることから、ここでは本内容を付録として掲載するにとどめます。

## 12.1 プログラミング

**プログラミング** とは、人間の意図した処理をコンピュータに行わせるために、一連の指示を記述することです。

コンピュータは、指示されたことを機械的に正しく処理するのは得意です。しかし一方で、コンピュータは人間と違って意思を持ちません。そこで人間が指示を与え、コンピュータはその通りに行動する、という一種の主従関係によって、コンピュータは計算などの処理をすることができるようになります。

ほとんどのプログラミングは、後述する「プログラム言語」（プログラミング言語と称することも多い）を用いて処理を記述します。以下、プログラム言語を用いて記述された一連の処理を、**プログラム** と呼びます。また一般的に、プログラミングを行う人（狭義には職業とする人）を「プログラマ」と呼びます。

## 12.2 プログラム言語

**プログラム言語** は上述の通り、人間がコンピュータに実行させたい処理を記述する言語のことで、プログラミング言語ともいいます。情報処理の分野では、人間が日常生活で読み書き（および会話）するための言語、例えば日本語や英語などを「自然言語」と称し、プログラム言語と明確に切り分けます。

計算機の中枢として各種の処理をこなす「CPU(中央演算装置)」は、俗に機械語（マシン語）で記述された処理を解読し、その記述内容にしたがって一連の処理を行います。しかしこの機械語は、人間が日常的に目にする自然言語や数式などと大きく異なる、人間にとって非常に読みにくいものです。よって、人間が機械語を直接記述する、というのは非常に難しい行為となります。また機械語は一般的に、計算機の基盤となるハードウェアやオペレーティングシステム（詳しくは別の講義で習ってください）の種類によって異なっています。よって機械語は、異なる計算機にわたって同一のプログラムを持ち運ぶ、というような利用方法を考えてみても、不便なものであることがわかります。

そこでプログラミングの現場においては、機械語よりも人間にとって読みやすい言語で処理を記述し、計算機が個々のハードウェアやオペレーティングシステムに対応する形でこれを解読し、機械語に翻訳することで実行可能な状態にする、というような仕組みが用いられているのが一般的です。最近では「プログラム言語」といえば、この「機械語よりも人間にとって読みやすい言語」を指すのが一般的になりつつあります。

プログラム言語には大きく **コンパイラ言語** と **インタープリタ言語** に分けられます。「機械語への翻訳」を実行前に済ませておく仕組みがコンパイラで、実行時に同時に行うのがインタープリタです。一般的には、後者のほうが手軽だが、前者のほうが処理が高速、という傾向にあります。なおコンパイラによる翻訳処理を、一般的に**コンパイル**と称します。

一言でプログラム言語にも多種多様なものがありますが、早稲田大学の理工系学部では、C, FORTRAN, Java の 3 種類のプログラム言語が 1 年生の履修科目となっています。またそれ以外に、BASIC というプログラム言語が有名かつ簡潔で、多くの現場で用いられています。以下、これらのプログラム言語について列挙します。

### C 言語

1973 年に AT&T ベル研究所のデニス・リッチーが主体となって作ったプログラム言語であり、現在でも大学の非常に多くの情報系学部・学科が履修科目等に採用しています。

UNIX をはじめとするオペレーティングシステムの記述にも用いられていることから、機械語に近いレベルの処理も記述できるのが特徴です。

C 言語はコンパイラ言語であり、実行速度やコンパイラ速度の面で他の言語より優れていますが、文法的な複雑さと、（ミスがあったときの）実行時の危険性の面が問題とされています。

現在では、C 言語に「オブジェクト指向」という考え方を導入した「C++言語」も活発に利用されています。

### FORTRAN 言語

1954 年に IBM のジョン・バッカスによって考案されました。コンピュータにおける史上最初のコンパイラ言語と言われています。

特に科学技術系の計算速度では2007 年現在でも他の言語を圧倒しており、よく使われ続けられています。物理学、機械工学や建築工学などの分野では、FORTRAN を用いた大規模な計算機シミュレーションが非常に多く用いられているため、これらの分野に関係する学科では FORTRAN 言語の教育が現在も活発です。※筆者の非常勤先大学の機械工学科、建築学科などは実際に FORTRAN 言語が履修科目に含まれていました。

### Java 言語

1990 年代前半にサン・マイクロシステムズ社でジェームズ・ゴスリンなどの人々によって開発された、比較的新しい言語です。ハードウェアやオペレーティングシステムの種類を問わず共通のプログラムを書ける、という点が特徴です。また「オブジェクト指向」という考え方を最も徹底的に導入したプログラム言語のひとつでもあり、そのためにソフトウェアの開発と保守の複雑さを低減し、開発効率と保守性を高めたという点も特徴です。

Java 言語は特に、1990 年代以降に発達した新しい情報技術の分野において、非常に急速に普及しています。例えば以下のような環境

- Web をはじめとするネットワーク環境上でのプログラム
- 携帯電話をはじめとする新しいハードウェア上でのプログラム
- 生命情報をはじめとする新しい学術分野でのプログラム

において、Java は特に標準的なプログラム言語として重宝されています。※筆者の非常勤先大学の生命情報系の学科では実際に Java 言語が履修科目に含まれていました。

### BASIC 言語

FORTRAN の文法をもとにして、1970 年代以降に広く普及された、インタープリタ型言語の代表的なものです。関数、ポインタ、構造体、といった複雑な概念を習得しなくてもある程度の処理を記述できることから、初心者向けのプログラム言語として知られています。

Windows 上のプログラム言語として現在も広く用いられている Visual Basic は、もとは BASIC 言語から派生するようにして開発されました。

# 13 【付録 B】初めての Linux 環境

本章では、筆者が担当した実習科目にて、大学に入学したばかりの 1 年生前期の学生に、プログラミングそのものの知識と並行して教える必要があった Linux の知識や操作方法について、必要最低限のものを列挙したものです。本書の主旨から外れる内容なので削除してもよかったのですが、とりあえず付録として掲載しています。あくまでも筆者の非常勤先大学の計算機環境を説明する章ですので、必ずしも読者の環境にて同一の操作ができるとは限らないと考えて下さい。

## 13.1 Linux とは

皆さんは大学の計算機の講義の中で、きっと**オペレーティングシステム**という単語を習うことでしょう。ここではごく簡単な説明にとどめますが、オペレーティングシステムとは、計算機を構成する各種装置（ハードウェア）に直接命令を送るプログラムのことであり、計算機の電源を入れたときにハードウェアが最初に読み込むプログラムと考えることもできます。

オペレーティングシステムの中でも有名で、皆さんの日常生活に浸透しているものといえば、Microsoft 社による **Windows** と、Apple 社による **Mac OS** です。家庭用パソコンのための商用オペレーティングシステムといえば、この 2 つが 10 年以上の長い間にわたって双璧として活躍しています。

それに対して、主に業務用や研究用の目的で、Windows や Mac OS よりも長い歴史を誇るオペレーティングシステムに **UNIX（ユニックス）**というものがあります。UNIX は 1970 年代から開発が進められているオペレーティングシステムで、主にワークステーションという（一般的にパソコンよりも高価な）計算機で活躍していました。

**Linux** は、UNIX を専用ワークステーションではなくパソコン用に開発し、しかも無料で配布する（現在では商用製品も多数存在しますが）という、画期的な試みの中で生まれたオペレーティングシステムの代表です。Linux の中枢部分は、1991 年に当時フィンランドのヘルシンキ大学在学中であったリーナス・トーバルズが個人で開発を開始したものであり、彼のファーストネームをとって Linux と呼ばれています。

Linux の発音は、リナックス、リーヌークス、リヌックス、リヌクス、ライナックス、リーナクス、などさまざまなものが聞かれます。筆者は **リナックス** と発音しています。

## 13.2 Linux システムにログインする

現在の大半のオペレーティングシステムでは、**マルチユーザ**という考え方を導入しています。マルチユーザとは簡単にいうと、複数の利用者が同一の計算機を共有する仕組みのことです。

マルチユーザなオペレーティングシステムでは、各利用者の設定や情報は、明確に区別されて保存されています。そのため利用者が計算機を使い始めるときには、自分が誰であるかを計算機に対して名乗り、それが計算機に認められて初めて、自分の設定や情報が計算機上で有効になり、計算機を利用できる状態になる、というような仕組みが搭載されています。この「自分が誰であるかを計算機に対して名乗る」という行為が**ログイン**というものです。

このログインという仕組みは、大学の端末室のような環境では非常に便利です。なぜなら皆さんは、大学の端末室では、いつも同じ席に座るとは限らないからです。マルチユーザなオペレーティングシステムでは、ログインの操作さえ正しくできれば、毎回違う席に座っていても、毎回同じ設定や情報を有効に活用できるようになります。ですので皆さんは、端末室でのプログラミング実習では、自分がどこに座っているかを意識する必要はありません。

ログインをするときに、利用者がキーボードを叩いて示す情報は、大抵の場合は以下の 2 つです。

**ユーザ名：** 利用者の名前のようなもの。他者にも公開されている。

**パスワード：** 利用者だけが知っている文字列。

ユーザ名とパスワードは、例えるなら銀行の口座番号と暗証番号のような関係にあります。口座番号は他者に知らせる情報であり、暗証番号は自分以外の誰にも知らせない情報です。

ですので今後、皆さんは端末室を利用するにあたり、

- **ユーザ名とパスワードを忘れないようにする**
- **パスワードを他者に知らせない**

という点を徹底するよう、くれぐれもお願いします。

なお今後、利用者の皆さんが計算機に向かって、キーボードやマウスを使って情報を与えることを、**入力する** と称します。逆に計算機が、利用者の皆さんに向かって、画面やスピーカやプリンタを使って情報を与えることを、**出力する** と称します。

以下に、ユーザ名とパスワードの入力手順を示します。

1. キーボードを叩くと、画面上のユーザ名を記入する欄に、皆さんの叩いた文字が表示されることを確認してください。[10]
2. 皆さんのユーザ名をキーボードで入力し、最後に「Enter」を押してください。この「Enter」を押す、という行為がないと、ユーザ名の入力は有効になりませんので、注意して下さい。
3. ここまで正常に操作できれば、画面上のパスワードの記入欄にキーボード入力できる状態になっているはずですので、確認してください。[11]
4. 皆さんのパスワードをキーボードで入力し、最後に「Enter」を押してください。この「Enter」を押す、という行為がないと、ユーザー名の入力は有効になりませんので、注意して下さい。また一般的に、パスワードは画面表示されません。[12]慣れないうちは、ついキーボード入力を間違えそうになるかと思いますが、注意して入力してください。
5. ユーザ名またはパスワードが間違っている場合には、それを警告する表示が出てくるはずです。正しく入力できた場合には、ユーザ名とパスワードの記入欄が消えて、新しい画面表示内容が見えてくるはずです。

どうしてもユーザ名やパスワードを入力できない場合、あるいはユーザ名やパスワードをまだ取得していない場合には、教員や TA をつかまえて、なんとかしてください。

## 13.3 ログイン後の操作とログアウト

ログインに成功すると、画面上の表示が大きく変わります。おそらく皆さんの画面には、**アイコン** と呼ばれるイラスト風の小さいボタンや、**メニュー** と呼ばれる選択肢が表示されていることでしょう。

このアイコンやメニューは、皆さんのマウス操作によって、計算機上の特定のプログラムを動かすために提供されているものです。このように、アイコンやメニューなどの視覚的な部品を使って、キーボードだけでなくマウスも使って計算機を操作する仕組みを、**GUI（グラフィカル・ユーザ・インタフェース）**といいます。また、このような操作によって起動される特定のプログラムを、**アプリケーション** といいます。また、画面上でアプリケーションが占有する、窓のような長方形領域を、**ウィンドウ** といいます。また、多数のウィンドウを操作する仕組みの整った GUI を、**ウィンドウシステム**といいます。

作業を終了して端末室を出るとき、皆さんは必ず、**ログアウト** という操作をしなければなりません。この操作は、計算機を「ログイン操作ができる状態」に戻すことを指します。

[10] もしユーザ名の記入結果が表示されない場合には、画面上の矢印記号（カーソル）がユーザ名の記入欄を指すように、マウスで移動させてください。そこでマウスの左ボタンを押す（クリックする）ことによって、ユーザ名の記入欄を使えるようになります。

[11] もしパスワードをキーボード入力できない場合には、画面上の矢印記号（カーソル）がパスワードの記入欄を指すように、マウスで移動させてください。そこでマウスの左ボタンを押す（クリックする）ことによって、パスワードの記入欄を使えるようになります。

[12] 多くのシステムでは、パスワード入力時に、「****」という伏せ字が表示されます。

皆さんがログアウト操作をしないで端末室を退出してしまうと、この計算機を他の人が使うことはできません。それどころか、悪意ある人が、皆さんになりすまして計算機を使ってしまい、皆さんが保存している大切な情報を失ってしまう可能性があります。これらのことから、

- **退出時にはログアウトを忘れない**

という点を、くれぐれも徹底して下さい。
ログアウトの方法は以下の通りです。

1. 画面左上の「アクション」をマウスでクリックする。
2. 選択肢が下に伸びるようにして一覧表示されるので、その中から「ログアウト」をマウスでクリックする。
3. 「本当にログアウトしてもよろしいですか？」という表示が出たら、アクション欄で「ログアウト」をマウスで選択して、「OK」をマウスでクリックする。

## 13.4 ターミナルとブラウザの起動

非常勤先大学にて筆者の担当科目を受講するにあたり、真っ先に覚えないといけないアプリケーションが 3 つありました。それはターミナル、ブラウザ、テキストエディタです。このうちテキストエディタについては 13.6 節で後述するとして、ここではターミナルとブラウザについて説明します。

### 13.4.1 ターミナル

**ターミナル**とは、キーボード入力によって計算機を操作するためのアプリケーションです。ターミナルは以下のような操作によって起動されます。

1. メニューの「アプリケーション」をマウスで選ぶ。
2. メニューの「システム・ツール」をマウスで選ぶ。
3. メニューの「GNOME 端末」をメニューで選ぶ。

以下の資料では、GNOME 端末を「ターミナル」と称します。

### 13.4.2 ブラウザ

**ブラウザ**とは、計算機ネットワークを通して発信されている情報を閲覧するためのアプリケーションです。最近では、俗に「ネットを使う」というと、ブラウザを通して情報を閲覧することを指す場合が多いようです。

ブラウザで表示するための情報発信の基盤となるネットワーク技術を、一般的に **World Wide Web** または単純に Web(ウェブ：クモの巣) と呼びます。ブラウザは正確には **Web ブラウザ**とも呼ばれます。

## 13.5 Linux 上での主要なコマンド

続いて、ターミナル上で入力できる Linux 主要なコマンドについて説明します。なお本書では、筆者が非常勤先大学の担当科目で実際に紹介した最小限のコマンドだけを説明します。これ以上の詳しい内容については、検索エンジンを用いて Web ブラウザ上で調べるか、専門書を購入するか、のいずれかにしてください。

なお以下に紹介するコマンドはすべて、単にその文字を入力するだけでなく**最後に Enter を押して初めて有効になる**という点に、くれぐれも注意して下さい。

### 13.5.1 ファイル

計算機ではデータや文書を **ファイル** という単位で保存します。おそらく皆さんは、何枚かの紙をひとまとめにする容器をファイルと呼んだことがあるかと思いますが、あれと同様に、ひとまとめのデータや文書を計算機ではファイルという単位で扱います。

これから作成される皆さんのプログラムも、各々が 1 個のファイルとして保存されます。

### 13.5.2 ディレクトリ（またはフォルダ）

いくつかのファイルをまとめて保管する入れ物を、**ディレクトリ**または**フォルダ** といいます。歴史的に Linux をはじめとする UNIX 系のオペレーティングシステムではディレクトリという単語が使われていますが、ウィンドウシステムではフォルダと呼ぶことが多いため、現在では両者は同義語として混在しています。原則としてディレクトリと呼ぶことにします。

ディレクトリは二重三重の入れ子を構成することができます。言い換えれば、ディレクトリの中にディレクトリを作ることが可能です。これは例えるなら、○○大学の中に△△学部があり、△△学部の中に□□学科があり、というような上下関係を作ることに類似しています。このような入れ子の構造、または上下関係をもつ構造を、**階層構造** と呼びます。

ターミナルを操作する際には常に、いま自分がどこのディレクトリの中にいるか、という状態を意識する必要があります。自分がいま所在するディレクトリを **カレントディレクトリ** といいます。ターミナル上でカレントディレクトリを表示するためには、

```
pwd
```

と入力してください。これでカレントディレクトリが表示されます。例えば

```
/home/itot
```

と表示されたとします。これは**「home というディレクトリの中に、itot というディレクトリがあって、自分はいま itot というディレクトリの中にいます」** ということを意味しています。

### 13.5.3 ディレクトリの作成

前節にて「ディレクトリの中にディレクトリを作ることが可能です」と書きました。そこで続いて、自分のカレントディレクトリの中に新しいディレクトリを作成する方法を説明します。いま、**「itot というカレントディレクトリの中に、java というディレクトリを作成したい」** とします。このようなときには、

```
mkdir java
```

と入力してみて下さい。ここで mkdir というコマンドは、カレントディレクトリの直下に、新しいディレクトリを作成する、という意味を持ちます。

### 13.5.4 ディレクトリの移動

続いて、カレントディレクトリを移動する方法について説明します。いま、**「itot というカレントディレクトリの中に java というディレクトリがあって、自分はいまそこに移動したい」** とします。このようなときには、

```
cd java
```

と入力してみて下さい。ここで cd というコマンドは、カレントディレクトリから、その直下にあるディレクトリに移動する、という意味を持ちます。上記のコマンドのあとに pwd コマンドを行うと、きっと

```
/home/itot/java
```

と表示されるのではないかと思います。

続いて、**「カレントディレクトリの 1 個上のディレクトリに移動したい」** とします。このようなときには、「1 個上のディレクトリ」を意味する..(ピリオド 2 個) という記号を使って

```
cd ..
```

と入力してください。上記のコマンドのあとに pwd コマンドを行うと、きっと

```
/home/itot
```

と表示されるのではないかと思います。

さて皆さんは、ログインをしてターミナルを起動すると、原則として毎回同じディレクトリから作業を開始することになるはずです。一般的にオペレーティングシステムでは、各々の利用者に対して、作業の開始場所となるディレクトリを指定することができます。この「作業の開始場所となるディレクトリ」を、**ホームディレクトリ**と呼びます。ターミナル上でホームディレクトリに戻る際には、

```
cd
```

と入力してください。cd の後に何も打たないで Enter を押すと、ホームディレクトリに戻る、と思っていただければと思います。

### 13.5.5　ディレクトリの中身を見る

演習が進むと、おそらくプログラムのファイルがたくさんつくられることと思います。自分のディレクトリの中に、どんなファイルが生成されているかを確認するには、

```
ls
```

というコマンドを入力してみて下さい。自分が作成した、末尾（拡張子）が java であるプログラムファイルの名前がそれに似た名前で拡張子が class であるファイルの名前が表示されるでしょう。本書に沿ってプログラミングを進めている途中であれば、例えば

```
Bmi.class Bmi.java Calculate.class Calculate.java
```

というように表示されるのではないかと思います。
なお、ファイルの大きさや更新日時などの付随情報も同時に表示したい場合には、

```
ls -l
```

というように、「-l」をつける必要があります。

### 13.5.6　ファイルのコピー

プログラミングの課題では、似たような複数のプログラムを作成することがあります。このとき各々のファイルを、1 行目から作成しなおすのは非効率ですよね。そこでプログラミングの多くの場面では、まず似たような既存のプログラムファイルをコピーして、そのコピーしたファイルを編集して新しいプログラムを作ります。
ファイルをコピーするとき、例えば Hello.java というファイルを Calculate.java というファイルにコピーするときには、

```
cp Hello.java Calculate.java
```

というように、cp というコマンドを使い、続いてコピー元ファイル名、コピー先ファイル名、という順に入力してください。ここで、コピー元ファイル名と、コピー先ファイル名の順番を逆にしないよう、くれぐれも注意して下さい。
また、上記のコマンドは Hello.java が置かれているディレクトリにて実行すれば有効になります。他のディレクトリをカレントディレクトリにしているときに上記のコマンドを実行する場合には、「パス」という考え方で Hello.java および Calculate.java の置き場所を指示する必要があります。本章では説明を割愛します。

### 13.5.7　ファイルの移動

ファイルのコピーに似たコマンドとして、移動というコマンドがあります。移動というよりは「ファイル名の変更」と考えたほうが、わかりやすいかもしれません。ファイルを移動するとき、例えば Hello.java

というファイルを Calculate.java というファイル名に変えるときには、

```
mv Hello.java Calculate.java
```

というように、mv というコマンドを使い、続いて移動元ファイル名、移動先ファイル名、という順に入力してください。ここでも、移動元ファイル名と、移動先ファイル名の順番を逆にしないよう、くれぐれも注意して下さい。

### 13.5.8 ファイルの削除

不要になったファイルを削除する方法について説明します。例えば Unnecessary.java というファイルを削除するには、

```
rm Unnecessary.java
```

というように、rm というコマンドを使い、続いて削除するファイル名を書いてください。なお、**rm コマンドによって削除されたファイルは復活不可能** ですので、rm コマンドを使うときは、くれぐれも慎重にお願いします。

### 13.5.9 Java のコンパイルと実行

Java 言語のプログラムを開発したら、コンパイルの必要があります。例えば Hello.java というファイルをコンパイルするには、

```
javac Hello.java
```

というように、javac コマンドを使ってください。javac のあとには、コンパイル対象となるファイル名を記載します。
カレントディレクトリにあるプログラムファイルを全てコンパイルするときには、

```
javac *.java
```

というように実行してください。ここで「*」とは「全ての」という意味をもつ記号です。このような記号を使ってファイル名などを表現することを、**正規表現** といいます。正規表現に用いる記号には「*」以外にもいくつかあります。詳しくは検索エンジンで調べるか、専門書で独習して下さい。
Java 言語で開発されたプログラムを実行するときには、

```
java Hello
```

というように、java コマンドを使ってください。ここで java コマンドの後の文字列には、main を持つクラスのクラス名、いいかえればプログラムファイル名から「.java」を除いたものを使います。

## 13.6 Emacs で文字情報を編集する

Java 言語のプログラムファイルをはじめとする文字情報を編集する際には、**テキストエディタ** というアプリケーションを用います。Windows では「メモ帳（Notepad）」が、Mac OS では「シンプルテキスト」が、パソコン購入時に付随されている簡易なテキストエディタの代表格といえるでしょう。

本章では「Emacs」というテキストエディタを紹介します。Emacs は、Windows でも Mac OS でも Linux でも共通に使えるテキストエディタの代表的なものであり、大学に限らず非常に多くの現場に普及しています。

Emacs を使って特定のファイルを編集するときには、ターミナル上で

```
emacs Hello.java
```

というように、emacs コマンドのあとにファイル名を書いて実行します。既に存在するファイルであれば、そのファイルの中身が表示されます。初めて編集するファイルであれば、白紙の状態で Emacs を起動します。

ただし、このコマンドで Emacs を実行すると、それを実行したターミナルに別のコマンドを入力できない状態になってしまいます。それを避ける方法として本講義では、

```
emacs Hello.java &
```

というように、**最後に&をつけて起動することを推奨します。**

ここまで出来ましたら、Emacs のウィンドウ上でプログラムを編集し、保存して下さい。保存の操作は以下の 2 通りのいずれかです。

- Ctrl キーと x キーを同時に押し、続いて Ctrl キーと s キーを同時に押す
- メニューで「ファイル」の「保存」を選ぶ

正しく保存できれば、**Emacs のウィンドウの左下端に「Wrote ...」**というメッセージが表示されます。これを確認できましたら、正常に保存できたと思ってください。

Emacs には、他にも非常に多くの機能があるのですが、ここでは最小限の解説にとどめます。これ以上の詳しい内容については、検索エンジンを用いて Web ブラウザ上で調べるか、専門書を購入するか、のいずれかにしてください。

## 13.7 困ったらここを読もう

筆者が非常勤講師としてプログラミングを実習するにあたり、勤務先大学の環境にて多くの学生から出てきた質問に対する回答を、以下にまとめてみました。

### [質問] ターミナルに何を入力しても反応しない・Emacs を起動するとき行末に&を入れるのを忘れた

Emacs を起動するとき、例えば

```
emacs Hello.java &
```

というように、文末に&を入れることを強く推奨しています。これを入れ忘れると、Emacs を終了するまでの間は、ターミナル（GNONE 端末）に何をキーボード入力しても受け付けてくれません。

もし Emacs の起動時に&を入れ忘れたら、ターミナルにて **Ctrl キーと z キーを同時に押し、続いて bg と入力して Enter キーを押す**という操作を試みてください。この操作によって、すでに起動していた Emacs は、文末に&をつけて起動したのと同じ状態になります。そして、ターミナルはキーボード入力を受け付けるようになります。

### [質問] プログラムを書き換えても javac の結果が変わらない・プログラムを書き換えて保存したつもりが保存されていなかった

プログラムを書き換えても、その後の javac コマンドや java コマンドの結果が変わらない場合、たいていは Emacs でファイル名を誤っているか、保存に失敗しています。

Emacs 上で書いたプログラムのファイル名が正しいかどうかを確認するには、まず **Emacs のウィンドウの左下部分、太字でファイル名が書かれている部位**を見てください。ここで書かれたファイル名が誤っている場合には、ファイル名を正してください。本章で説明した mv コマンドが有効でしょう。

Emacs 上で書いたプログラムが保存されているかどうかを確認するためには **Emacs のウィンドウの左下部分、ファイル名の左**を見てください。ここに** という印が見える場合には、プログラムは最新の状態で保存されていませんので、もう一度保存してください。

### [質問] Emacs が書き込みを受け付けなくなった

Emacs の操作中に、何らかのキー操作をしてしまったために、プログラム本文の書き込みを受け付けなくなることがあります。このような場合には、**Emacs のウィンドウの一番下の行を見てください。**ここに何らかのメッセージがある場合には、これに応じた入力を要求されていて、それを入力するまでプログラム本文を書き込めない状態になっています。これを元の状態に戻す一番簡単な方法は、**Emacs のウィンドウの一番下の行に、マウス操作でカーソルを表示させて、そこで Ctrl キーと g キーを同時に押す**という操作を行うことです。Ctrl キーと g キーを同時に押すことで、直前の操作を中止することができるため、これによって Emacs はプログラムを書き込める状態に戻ります。

### [質問] プログラムが反復処理し続けて終了しない

特に for 文や while 文を含むプログラムを書いたときに、プログラムに誤りがあると、実行時に反復処理を無限に繰り返して終了しないことがあります。このようなときは、ターミナルにて、**Ctrl キーと c キーを同時に押す**　という操作を試みてください。この操作によって、実行中のプログラムが強制終了されます。

なお、この操作は自分で開発したプログラムに限らず、どのようなプログラムでも同様に使うことができます。

### [質問] さっきターミナルに入力した命令を思い出したい

ターミナル上で、上向き、または下向きのカーソル（矢印）キーを押してください。1 回押すごとに、前回の命令を表示します。何回か押すと、何回か前（または後）の命令を表示します。この状態で Enter キーを押すことで、表示された命令をもう一度実行できます。

# 索 引

## 参考文献

[1] はじめての Java プログラミング改訂版基本編，秀和システム，ISBN4-7980-0936-9.

[2] Java 1 はじめてみようプログラミング，三谷純，翔泳社，ISBN-13-978-4798120980.

[3] http://ja.wikipedia.org/wiki/

[4] http://www.ics.kagoshima-u.ac.jp/edu/proen3/

[5] http://www.tohoho-web.com/java/

## 著者略歴

■伊藤貴之（いとう　たかゆき）

1992 年　早稲田大学大学院理工学研究科修士課程修了
　　日本アイ・ビー・エム株式会社東京基礎研究所研究員を経て，
現　　在　お茶の水女子大学教授
　　大学院人間文化創成科学研究科　自然・応用科学系
　　理学部情報科学科
　　博士（工学）
　　専門：コンピュータグラフィックス，可視化

Java プログラミング入門

2013 年 10 月 25 日　発行

著　者　伊藤 貴之
発　行　お茶の水女子大学附属図書館(E-book サービス)
　　〒112－8610　東京都文京区大塚 2－1－1
　　http://www.lib.ocha.ac.jp/
　　電話　03-5978-5835　FAX 03-5978-5849
　　ISBN978－4－904793－04－6