第 1 章

画面に文字を表示しよう

画面に文字を表示するプログラムを通じて、Java に慣れましょう。

- □ コメント
- □ 文
- □ 画面への表示とストリーム
- □ 文字列リテラル
- □ 改行

▶『明解Java入門編』List 1-1 (p.6)

問題1-1

コンソール画面に『初めてのJavaプログラム。』と『画面に出力しています。』とを、連続して一行ずつ表示するプログラムを作成せよ。

```
// 画面への出力を行うプログラム

class Hello {

   public static void main(String[] args) {
      System.out.println("初めてのJavaプログラム。");
      System.out.println("画面に出力しています。");
   }
}
```

注意!! 数字の1ではなく小文字のエルです。

実行結果

```
初めてのJavaプログラム。
画面に出力しています。
```

ソースプログラムとソースファイル

コンソール画面に『初めてのJavaプログラム。』『画面に出力しています。』と表示するプログラムです。エディタなどを用いて、ここに示すとおりに打ち込みましょう。

▶ 大文字と小文字は区別されます。余白や " などの記号を全角文字で打ち込んではいけません。なお、余白の部分は、スペース・タブ・リターン（エンター）のキーを使って打ち込みます。

{ } や ; などの記号文字の読み方は、p.9 の **Table 1-1** にまとめています。

私たちが《文字の並び》として作るプログラムを**ソースプログラム**（*source program*）、ソースプログラムを格納したファイルを**ソースファイル**（*source file*）と呼びます。

▶ sourceは、『もとになるもの』という意味です。

ソースファイルの名前は、**class**の後ろに書かれた**クラス**（*class*）の名前（本プログラムでは*Hello*）に拡張子 .javaを加えたものとするのが原則です（**Fig.1-1**）。打ち込み終わったら、Hello.javaというファイル名で保存します。

▶ プログラムの保存先については、p.14 で解説しています。

ソースプログラムのコンパイルとクラスファイル

ソースプログラムは、そのままでは実行できません。**バイトコード**（*bytecode*）と呼ばれる実行可能な形式に変換するための作業である**コンパイル**（*compile*）が必要です。本プログラムの場合、コンパイルは以下のように行います。

```
▶ javac Hello.java↵
```

ここで、**拡張子 .java を省略することはできません。**

コンパイルが完了すると、Hello.classというファイルが生成されます。これは、**クラスファイル**（*class file*）と呼ばれ、その中身はバイトコードです。

ソースプログラムに綴り間違いなどがあると、コンパイル時にエラーが発生して、その旨のメッセージが表示されます。その際は、プログラムをよく読み直して、ミスを取り除いた上で、再度コンパイルの作業を試みましょう。

プログラム（クラス）の実行

クラスファイルからクラスを読み込んで実行するのが java コマンドです。クラス *Hello* の実行手順は、次のようになります。

```
▶ java Hello↵
```

ここで、**拡張子 .class を付けてはいけません**（ここで指定するのは《クラス》の名前であって、《クラスファイル》の名前ではないからです）。

プログラムを実行すると、コンソール画面への出力が行われます。

▶ 本書では、実行結果をプログラムリストの枠内に示しています。

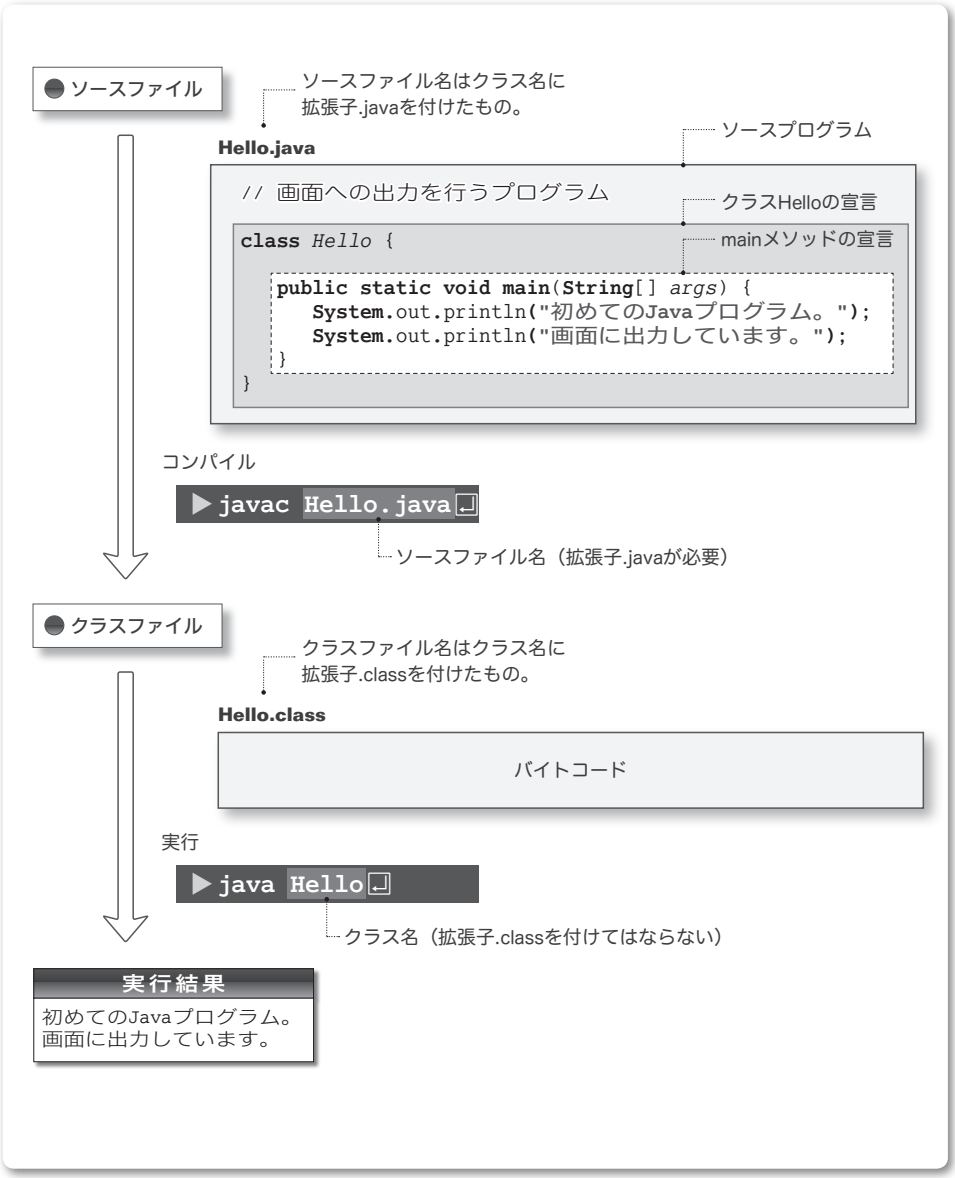

Fig.1-1 プログラムのコンパイルと実行

コメント（注釈）

プログラム先頭の連続する２個のスラッシュ記号 // は、『**この行のこれ以降は、プログラムの《読み手》に伝えることです。**』という表明です。すなわち、プログラムそのものではなく、プログラムに対する**コメント**（*comment*）すなわち**注釈**です。

コメントの有無やその内容は、プログラムの動作に影響を与えません。作成者自身を含めて、プログラムの読み手に伝えたいことがらを、簡潔な言葉（日本語や英語など）で記述するようにします。

他人が作成したプログラムに適切なコメントが書かれていれば、読むときに理解しやすくなります。また、自分が作ったプログラムのすべてを永遠に記憶することなど不可能ですから、コメントの記入は作成者自身にとっても重要なことです。

＊

コメントの記述法には３種類があり、自由に使い分けられるようになっています。

a 伝統的コメント（traditional comment）

```
/*
   伝統的コメント
*/
```

注釈を /* と */ で囲みます。開始の /* と終了の */ とが**同一行になくてもよい**ため、複数行にわたるコメントの記述に効果的です。

▶ 《伝統的》という名称は、Ｃ言語のコメントと同じ形式であること（1970 年代から使われていること）に由来します。コメントを閉じるための */ を、/* と書き間違えたり、書き忘れたりしないように注意しましょう（この点ではbの記述法も同様です）。

b 文書化コメント（documentation comment）

```
/**
   文書化コメント
*/
```

注釈を /** と */ で囲みます。aと同様、複数行にわたることができます。

▶ この形式のコメントから、プログラムの仕様書となるドキュメント（文書）を生成できます。第 13 章で解説します。

c 行末コメント（end of line comment）

```
// 行末コメント
```

// から、その行の末端までがコメントとなります。複数行にわたることができない反面、手短なコメントの記述に便利です。

▶ 文書化コメントと伝統的コメントは、入れ子にする（コメントの中にコメントを入れる）ことができません。そのため、以下のコメントは、コンパイル時にエラーとなります。

```
/**  /* このようなコメントは駄目 !! */   */
```

最初の */ がコメントの終了とみなされ、後ろ側の */ はコメントとはみなされないからです。

ただし、文書化コメントと伝統的コメントの中では // を使えますし、その逆もＯＫです（特別扱いされずに、注釈として書かれた文字であるとみなされます）。そのため、以下に示すのは、いずれも正しいコメントであり、エラーにはなりません。

```
/* // このコメントはＯＫ !! */
// /* このコメントもＯＫ !! */
```

プログラムの構造

コメント以外のプログラム本体部分を理解していきましょう。本プログラム *Hello* は、**Fig.1-2** に示す構造となっています。

Fig.1-2 プログラムHelloの構造

■ クラス宣言

Aの部分はプログラム全体の《骨組み》です。少し難しい言葉で説明すると、『**名前が *Hello* であるクラス**（*class*）**のクラス宣言**（*class declaration*）』です。

もっとも、その詳細を今すぐ理解する必要はありません。現時点では、右の形式で書くものと覚えておけば十分です。

```
class クラス名 {
   // ...
}
```

本プログラムの《クラス名》は *Hello* です。このように、クラス名の先頭文字は、**大文字**とするのが原則です。

なお、**ソースファイルの名前は、大文字・小文字の区別を含めてクラス名と同一でなければなりません**。たとえば、クラス名が *Abc* で、ファイル名が abc.java だと、コンパイルには成功しますが、**実行に失敗します**。

■ main メソッド

Bの部分は **main メソッド**（*main method*）の宣言です。

public static void や（**String**[] *args*）の部分は、後の章で学習します。それまでは、この部分を《決まり文句》として覚えておきましょう。

■ 文

プログラムを起動して実行すると、**main** メソッド中の**文**（*statement*）が、順次実行されることになっています。**文はプログラム実行の単位です**。

したがって、まず①の文が実行され、それから②の文が実行されます。いずれも、コンソール画面への表示を行います（これらの文の詳細は次ページで学習します）。

▶ メソッドについては、第 7 章以降で学習します。

▶『明解Java入門編』演習1-1(p.13)

問題1-2

プログラム中の文の終端を示すセミコロン；が欠如しているとどうなるか。プログラムをコンパイルして検証せよ。

```
// 画面への出力を行うプログラム（誤り：セミコロンが欠如）

class HelloError {

   public static void main(String[] args) {
      System.out.println("初めてのJavaプログラム。")
      System.out.println("画面に出力しています。")
   }
}
```

実行結果

コンパイルエラーとなるため実行できません。

文

前問のプログラム *Hello* 中の **main** メソッドには二つの文がありました。いずれも末尾はセミコロン；です。日本語の文の末尾に句点 。があるのと同様に、Java の文の末尾には原則としてセミコロン；が必要です（p.121）。本プログラム *HelloError* のように、必要なセミコロンが欠如すると、**コンパイルエラー**となります。

文字列リテラル

コンソール画面への出力を行う文（本問のプログラム *HelloError* の文ではなく、前問のプログラム *Hello* の文）を理解していきましょう。

まずは、"初めてのJavaプログラム。" と "画面に出力しています。" の部分に着目します。二重引用符 " で囲んだ文字の並びを、**文字列リテラル**（*string literal*）と呼びます。

リテラルとは、『文字どおりの』という意味です。たとえば、文字列リテラル "ABC" は3個の文字 A と B と C の並びを表します（**Fig.1-3**）。

コンソール画面への出力とストリーム

コンソール画面を含めて、外部との入出力には、**ストリーム**（*stream*）を利用します。ストリームとは、文字が流れる川のようなものです（**Fig.1-4**）。

System.out は、コンソール画面と結び付くストリームであって、**標準出力ストリーム**（*standard output stream*）と呼ばれます。

それに続く println は、( ) 中の内容（本図では、文字列リテラル "ABC"）をコンソール画面に表示した上で**改行する**（改行文字を出力する）働きをもつプログラムの《部品》です。このような部品のことを**メソッド**（*method*）と呼びます。

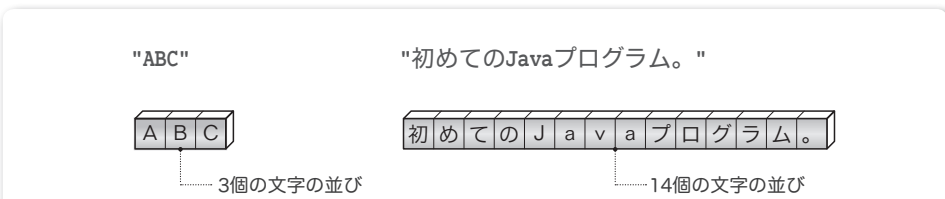

Fig.1-3 文字列リテラル

プログラム *Hello* では、まず『初めての Java プログラム。』が表示され、それから『画面に出力しています。』が次の行に表示されます。

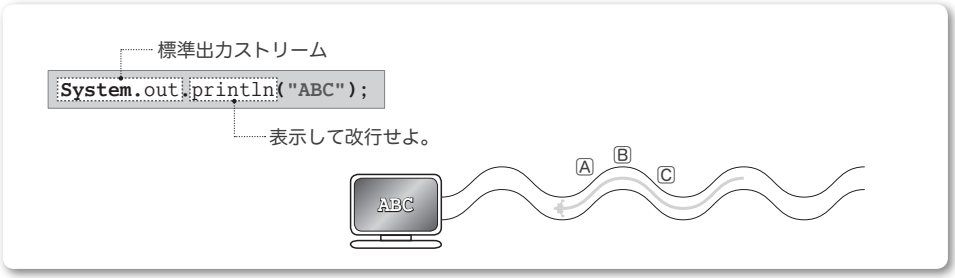

Fig.1-4 コンソール画面への出力

コメントアウト

プログラムの開発時に、『この部分が間違っているかもしれない。もしこの部分がなかったら、実行時の挙動はどう変化するだろうか。』と試しながらプログラムを修正することがあります。その際に、プログラムの該当部を削除してしまうと、もとに戻すのが大変な作業となります。

そこで、よく使われるのが**コメントアウト**という手法です。コメントとしてではなく、**プログラムとして記述されている部分を、コメントにしてしまうのです。**

プログラム *Hello* を以下のように書きかえて実行してみましょう。網かけ部がコメントとみなされますから、『初めての Java プログラム。』は表示されなくなります。

```
class Hello {
   public static void main(String[] args) {
//     System.out.println("初めてのJavaプログラム。");
       System.out.println("画面に出力しています。");
   }
}
```

行の先頭に 2 個のスラッシュ記号 // を書くだけで、その行全体をコメントアウトできるわけです。プログラムをもとに戻すのも簡単です。// を消すだけです。

なお、複数行にわたってコメントアウトする際は、以下に示すように /* … */ 形式を使うとよいでしょう。

```
class Hello {
   public static void main(String[] args) {
/*
       System.out.println("初めてのJavaプログラム。");
       System.out.println("画面に出力しています。");
*/
   }
}
```

コメントアウトされたプログラムは、読み手にとって紛らわしく、誤解されやすいものとなります（コメント化の根拠が、その部分が不要になったためなのか、何らかのテストを目的とするものなのか、などが分かりませんから）。コメントアウトの手法は、あくまでもその場しのぎのための一時的な手段と割り切って使いましょう。

問題1-3

『初めてのJavaプログラム。』と『画面に出力しています。』を、改行することなく連続して表示するプログラムを作成せよ。

```
// 画面への連続表示（その１）

class Hello1A {

   public static void main(String[] args) {
      System.out.println("初めてのJavaプログラム。画面に出力しています。");
   }
}
```

実行結果

```
初めてのJavaプログラム。画面に出力しています。
```

```
// 画面への連続表示（その２）

class Hello1B {

   public static void main(String[] args) {
      System.out.print("初めてのJavaプログラム。");
      System.out.println("画面に出力しています。");
   }
}
```

```
// 画面への連続表示（その３）

class Hello1C {

   public static void main(String[] args) {
      System.out.println("初めてのJavaプログラム。" + "画面に出力しています。");
   }
}
```

System.out.println と System.out.print

三つの解答プログラムを示しています。まずは、最初の二つを理解しましょう。

▶ プログラム *Hello1A* にのみ実行結果を示していますが、他のプログラムも同じ実行結果が得られます。

■ Hello1A

文字列リテラル "初めてのJavaプログラム。画面に出力しています。" を出力します。出力に利用しているのは、**System.**out.println メソッドです。

■ Hello1B

System.out.print メソッドと **System.**out.println メソッドの二つを使って画面への表示を行っています。

println の ln は、行（ぎょう）という意味の line の略です。println から ln を取り除いた print では表示後に**改行されません**。

System.out.print による『初めてのJavaプログラム。』の表示の後で改行されないため、『画面に出力しています。』は、同じ行に続けて表示されることになります。

文字列の連結

複数の文字列リテラルを + で結んだら、それらの文字列が連結された文字列が生成されることになっています。たとえば、`"ABC" + "DEF"` は、`"ABCDEF"` となります。

文字列の連結を利用して表示を行っているのが、*Hello1C* のプログラムです。

記号文字の読み方

Java のプログラムで利用する記号文字の読み方を **Table 1-1** に示します。

▶ **注意**：日本語版の MS-Windows などでは、**逆斜線（バックスラッシュ）** の代わりに**円記号 ¥** を使います。たとえば、次ページのプログラム *PrintName1B* の表示を行う箇所は、以下のようになります。

```
System.out.println("柴¥n田¥n望¥n洋");
```

みなさんの環境に応じて、必要ならば読みかえるようにしてください。

Table 1-1 記号文字の読み方

記号	読み方
+	プラス符号　正符号　プラス　たす
-	マイナス符号　負符号　ハイフン　マイナス　ひく
*	アステリスク　アスタリスク　アスター　かけ　こめ　ほし
/	スラッシュ　スラ　わる
\	逆斜線　バックスラッシュ　バック　※ JIS コードでは ¥
¥	円記号　円　円マーク　注意!!
$	ドル　ダラー
%	パーセント
.	ピリオド　小数点文字　ドット　てん
,	コンマ　カンマ
:	コロン　ダブルドット
;	セミコロン
'	一重引用符　単一引用符　引用符　シングルクォーテーション
"	二重引用符　ダブルクォーテーション
?	疑問符　はてな　クエッション　クエスチョン
!	感嘆符　エクスクラメーション　びっくりマーク　びっくり　ノット
&	アンド　アンパサンド
~	チルダ　なみ　にょろ　※ JIS コードでは ‾（オーバライン）
‾	オーバライン　上線　アッパライン
^	アクサンシルコンフレックス　ハット
#	シャープ　ナンバー
_	下線　アンダライン　アンダバー　アンダスコア
=	等号　イクオール　イコール
|	縦線
( )	左(右)括弧　左(右)丸括弧　左(右)小括弧　パーレン
{ }	左(右)波括弧　左(右)中括弧　ブレイス
[ ]	左(右)角括弧　左(右)大括弧　ブラケット
< >	小(大)なり　左（右)向き不等号

▶『明解Java入門編』演習1-2(p.20)

問題1-4

各行に１文字ずつ自分の名前を表示するプログラムを作成せよ。

```
// 自分の名前を１行に１文字ずつ表示（その１）

class PrintName1A {
   public static void main(String[] args) {
      System.out.println("柴");
      System.out.println("田");
      System.out.println("望");
      System.out.println("洋");
   }
}
```

実行結果

```
柴
田
望
洋
```

```
// 自分の名前を１行に１文字ずつ表示（その２）

class PrintName1B {
   public static void main(String[] args) {
      System.out.println("柴\n田\n望\n洋");
   }
}
```

改行

名前を各行に 1 文字ずつ表示するプログラムです。ここでは二つの解答プログラムを示しています。

▶ いずれも監著者の名前『柴田望洋』を表示するプログラムです。みなさんは、自分の名前に書きかえてください。

■ PrintName1A

四つの文字列リテラル "柴"、"田"、"望"、"洋" を **System.**out.println メソッドによって 1 個ずつ表示しています。println メソッドによる出力で改行文字が付加されることを利用しています。

■ PrintName1B

単一の **System.**out.println メソッドによって出力を行っています。文字列リテラルに埋め込まれている **\n** は、《改行文字》を表す特別な表記です。改行文字を出力すると、それに続く表示は、次の行の先頭から行われます（画面に \n と表示されるわけではありません）。

▶ 二つの文字 \ と n で構成される **\n** が表すのは、《改行文字》という**単一の文字**です。このように、目に見える文字として表記が不可能あるいは困難な文字は、逆斜線 \ で始まる**拡張表記**で表します。拡張表記の詳細は、第５章で学習します。

そのため、"柴"、"田"、"望" の各文字が表示された後に改行文字が出力されることになります。

▶ println メソッドによって出力していますから、最後の文字 "洋" の後ろには **\n** が不要です。なお、println メソッドでなく print メソッドによって出力するのであれば、"洋" の後ろにも **\n** が必要となります。

▶『明解Java入門編』演習1-3(p.20)

問題1-5

各行に1文字ずつ自分の名前を表示するプログラムを作成せよ。なお、姓と名の間は1行あけること。

```
// 自分の姓と名を1行に1文字ずつ表示（その1）

class PrintName2A {
   public static void main(String[] args) {
      System.out.println("柴");
      System.out.println("田");
      System.out.println();
      System.out.println("望");
      System.out.println("洋");
   }
}
```

実行結果

```
柴
田

望
洋
```

```
// 自分の姓と名を1行に1文字ずつ表示（その2）

class PrintName2B {
   public static void main(String[] args) {
      System.out.println("柴\n田\n\n望\n洋");
   }
}
```

空の行の出力

本問は、1行あける（空の行を出力する）方法を問う問題です。二つの解答を示しています。

■ PrintName2A

System.out.println による出力では、**( )** の中を空（から）にできることになっています。そのため、以下の文を実行すると、文字が表示されることなく改行だけが行われます（改行文字だけが出力されます）。

```
System.out.println();              // 改行する（改行文字を出力する）
```

本プログラムでは、姓を表示した後に、この文によって改行文字を出力しています。これで、1行あくことになります。

▶ **()** の中を空にできるのは println だけです。print では **()** の中を空にすることはできません。

■ PrintName2B

姓と名の間に、改行文字を表す拡張表記を2個並べています（網かけ部）。姓を表示した後に、改行文字が2個出力されるため、姓と名との間が1行あくことになります。

▶ どちらのプログラムでも、姓の最後の文字である "田" を出力した後に改行されて、さらにもう一度改行されることになります。

Java の特徴

Java は、利用者数が増え続けているプログラミング言語です。米国 Sun Microsystems 社によって開発され、1995 年 5 月の SunWorld で発表されました。ここでは、Java の特徴を簡単に紹介します。

▶ ここでは、『入門編』である本書で学習しない項目についても触れています。

▪ 無料で提供される

プログラミング言語を用いてプログラムを開発するためには、その言語用の開発ツールが必要です。Java の開発ツールは**無料で**提供されます。

▪ いったん作れば、どこでも実行できる … Write Once, Run Anywhere.

一般に、プログラミング言語で作成したプログラムは、特定の機器や環境でのみ動作するものとなります。Java で作成したプログラムは、（Java が動作する環境でありさえすれば）どこででも動きます。MS-Windows 用、Mac 用、Linux 用に別々にプログラムを作る、といったことは不要です。

▪ C 言語や C++ に似た構文

プログラミングで利用する語句や文の構造などの文法体系は、各言語で独自に決められています。Java の文法体系は、C 言語や C++ を参考にして作られていますので、それらの言語の経験者は、比較的容易に Java へ移行できます。

▪ 強い型付け

プログラムでは、整数・実数（浮動小数点数）・文字・文字列など、数多くのデータ型を扱います。各種の演算において、許されないもの・曖昧なものは、Java の開発ツールによって厳密にチェックされますので、信頼性の高いプログラムを作りやすくなります。

▪ オブジェクト指向プログラミングのサポート

クラスによるカプセル化・継承・多相性といった、**オブジェクト指向プログラミング**（*object oriented programming*）を実現するための技術がサポートされています。品質の高いソフトウェアを、効率よく開発できます。

▪ 無数のライブラリ

画面への文字表示・図形の描画・ネットワークの制御などのプログラムのすべてを自分で作ることは、現実的に不可能です。Java では、そのような機能の基本部分が、API（プログラムの部品の形態）のライブラリ（部品の集まり）として提供されています。API を利用すれば、目的とする処理を行うのは、簡単です。多方面にわたる多機能なライブラリが数多く提供されます。

▪ ガーベジコレクションによる記憶管理

多くのプログラミング言語では、オブジェクト（値を表すための変数のようなもの）を必要になった時点で生成できるようになっています。その一方で、『不要になってしまったオブジェクトの解放』の管理には、細心の注意が要求されます。Java では、オブジェクトの解放処理が自動的に行われますから、オブジェクトの管理が楽になります。

▪ 例外処理

予期せぬエラーなどの例外的な状況に遭遇したときの処理を、スマートに行えるようになっています。頑丈なプログラムの開発が容易です。

▪ 並行処理

一つのプログラム内で、複数の処理を同時並行的に実行できます。たとえば、画面に表示を行いながら別の計算を行う、といったことができます。

▪ パッケージによるクラスの分類

私たちが利用するディスク上のファイルは、ディレクトリ（フォルダ）ごとに分類して管理します。それと同じような感じで、Java のクラス（データと手続きをまとめたプログラムの部品）を、パッケージごとに分類できるようになっています。膨大な数におよぶクラスを効率よく管理できます。

Java の発展

Java は、頻繁にバージョンアップ（改訂）を重ねています。主要なバージョンの一覧を示したのが **Table 1-2** です。なお、バージョン 5.0 と 6 は、見かけ上のバージョン番号であって、内部バージョン番号は 1.5 と 1.6 です。本書で学習するプログラムは、すべて 5.0 以上で動作するものです。

▶ 内部バージョン 1.2 から 1.5 までは Java 2 という名称が用いられていましたが、1.6 から再び Java に戻っています。

Table 1-2 Javaの主要なバージョン

バージョン		コード名	リリース年月
JDK 1.0		—	1996 / 1
JDK 1.1		—	1997 / 2
J2SE 1.2		Playground	1998 / 12
J2SE 1.3		Kestrel	2000 / 5
J2SE 1.4.0		Merlin	2002 / 2
J2SE 5.0	(1.5)	Tiger	2004 / 9
Java SE 6	(1.6)	Mustang	2006 / 12

Java 開発キット

Java を使ってプログラムを開発するために提供されているのが、**Java 開発キット**（*Java Development Kit*）＝ **JDK** です。

Java 開発キットは、インターネット上のホームページから**無料で**ダウンロードできます。ダウンロードの方法やインストール（設置導入）の方法などは、以下のホームページで詳細に解説していますので、こちらをご覧ください。

http://www.bohyoh.com/　柴田望洋後援会オフィシャルホームページ

Java 用語辞典や Java に関する FAQ（よく聞かれる質問と回答を集めたもの）など、豊富なコンテンツを提供しています。

ソースプログラムとディレクトリ

本書で学習する数多くのソースプログラムを単一のディレクトリ（フォルダ）で管理することは、現実的ではありません。ディレクトリとファイルは、**Fig.1-5** のように構成しましょう。

お使いのシステムが MS-Windows であれば、ハードディスクに TokinagaraManabuJava ディレクトリを作り、その中に各章用のディレクトリ Chap01，Chap02，… を作ります。そして、各章用のディレクトリの中にソースプログラムを保存します。

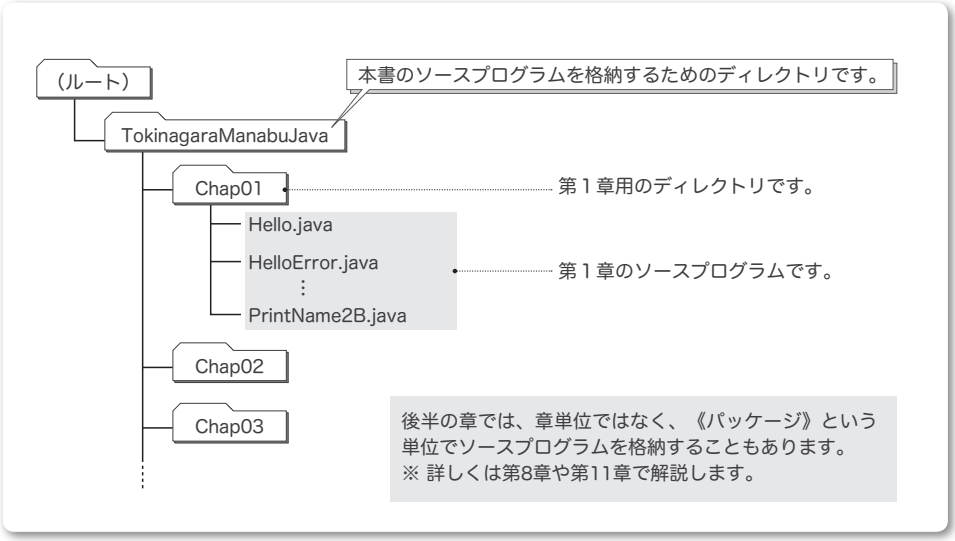

Fig.1-5　本書のソースプログラムのディレクトリ構成（一例）

▶　膨大な数のファイルを一元的に管理するのは困難です。そのため、Linux や MS-Windows などの OS（オペレーティングシステム＝基本ソフトウェア）では、階層構造をもつディレクトリ（フォルダ）によってファイルを管理します。

多数のディレクトリの中で、現在着目している（作業をしている）ディレクトリのことを**カレントディレクトリ**（あるいは**ワーキングディレクトリ**）と呼びます。

＊

Java プログラムのコンパイル・実行を行う際は、対象とするファイルが置かれているディレクトリをカレントディレクトリとするのが基本です。

そのため、プログラムのコンパイルをする前に、各章用のディレクトリにカレントディレクトリを移動する必要があります。カレントディレクトリの移動に利用するのが cd コマンドです。

```
▶ cd /TokinagaraManabuJava/Chap01↵
```

なお、MS-Windows で複数台のハードディスクがある場合は、ドライブの移動も必要です。もし TokinagaraManabuJava ディレクトリを D ドライブに作成しているのであれば、上のコマンドを実行する前に、次のコマンドを実行してカレントドライブを移動します。

```
▶ d:↵
```

※ディレクトリとファイルを区切る記号は OS によって異なります。多くの環境では /，\，¥ のいずれかです。本書は / で表記します。

錬成問題

※ 空欄部を埋めてください（次章以降の錬成問題も同様です）。

▪ Java は、(1) 指向プログラミングをサポートする言語である。Java プログラム開発に利用するツールの、アルファベット 3 文字の略称は (2) である。

▪ 私たち人間が文字の並びとして作成するプログラムは (3) と呼ばれ、それを格納したファイルには (4) という拡張子を与える。(3) はそのままでは実行できないので、(5) の作業が必要であり、それを行うのが (6) コマンドである。

(5) の結果生成されるバイトコードを格納したファイルは (7) と呼ばれ、その拡張子は (8) である。生成されたファイル内に収められている (9) を実行するのが (10) コマンドである。

▪ 作成者を含め、その読み手に伝えたいことがらを、簡潔な言葉としてプログラムに書き込まれたものが、(11) である。その有無や内容によってプログラムの動作が変わることは (12) 。

『伝統的 (11) 』は注釈を (13) と (14) とで囲み、『文書化 (11) 』は注釈を (15) と (16) とで囲む。いずれも開始の記号と終了の記号とが同一行の中に位置 (17) 。『行末 (11) 』は (18) からその行の末端までが注釈となる。この形式は、複数行にわたることが (19) 。

▶ (12) の選択肢：(a)ある　(b)ない
▶ (17) の選択肢：(a)しなければならない　(b)しなくてもよい
▶ (19) の選択肢：(a)できる　(b)できない

▪ "ABC" のように二重引用符 " で囲んだ文字の並びを、(20) と呼ぶ。

▪ `System.out.print("AB\nC" + "D\n\nEFG");` を実行すると (21) と表示される。

▪ 以下に示すのは、『風林火山』の各文字を 1 行に 1 文字ずつ表示するプログラムである。

```
(22) WindWoodFireMountain {
   public (23) void (24) ( (25) [] args) {
       (26) .out. (27) ("風");
       (26) .out. (27) ("林");
       (26) .out. (27) ("火");
       (26) .out. (27) ("山");
   }
}
```

▪ 以下に示す記号文字の読み方をカタカナで示せ。

; … (28)　　: … (29)　　. … (30)　　, … (31)

{ … (32)　　(… (33)　　' … (34)　　" … (35)