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。

- ソースプログラムとソースファイルとクラスファイル
- プログラムのコンパイルと実行
- クラス宣言
- main メソッド
- 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

問題1–1

▶『新・明解Java入門』List 1-1（p.6）

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“イチ”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

コンソール画面への表示を行うプログラムです。

▪ **ソースプログラム**（*source program*）… **人間が“文字の並び”として作るプログラム**

▶ 大文字と小文字は区別されます。" などの記号を全角文字で打ち込んではなりません。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、**Table 1-1**（p.7）にまとめています。

▪ **ソースファイル**（*source file*）… **ソースプログラムを格納したファイル**

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 を加えたものとします（**Fig.1-1**）。

▶ source は、『もとになるもの』という意味です。ソースプログラムの保存場所については、p.10 で学習します。

ソースプログラムのコンパイルとクラスファイル

そのままでは実行できないソースプログラムを、実行可能な**バイトコード**（*bytecode*）と呼ばれる形式に変換する作業が**コンパイル**（*compile*）です。本プログラムの場合、以下のように行います。なお、**拡張子 .java は省略できません。**

```
▶ javac Hello.java↵
```

【コンパイル】Hello.java をコンパイル

コンパイルが完了すると、Hello.class という名前の**クラスファイル**（*class file*）が生成されます。

▶ プログラムに綴（つづ）り間違いなどがあると、コンパイル時にエラーが発生します（メッセージが表示されます）。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

クラスファイル中の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* の実行は、以下のように行います。**拡張子 .class を付けてはなりません。**

```
▶ java Hello↵
```

【実行】クラス *Hello* を実行

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。

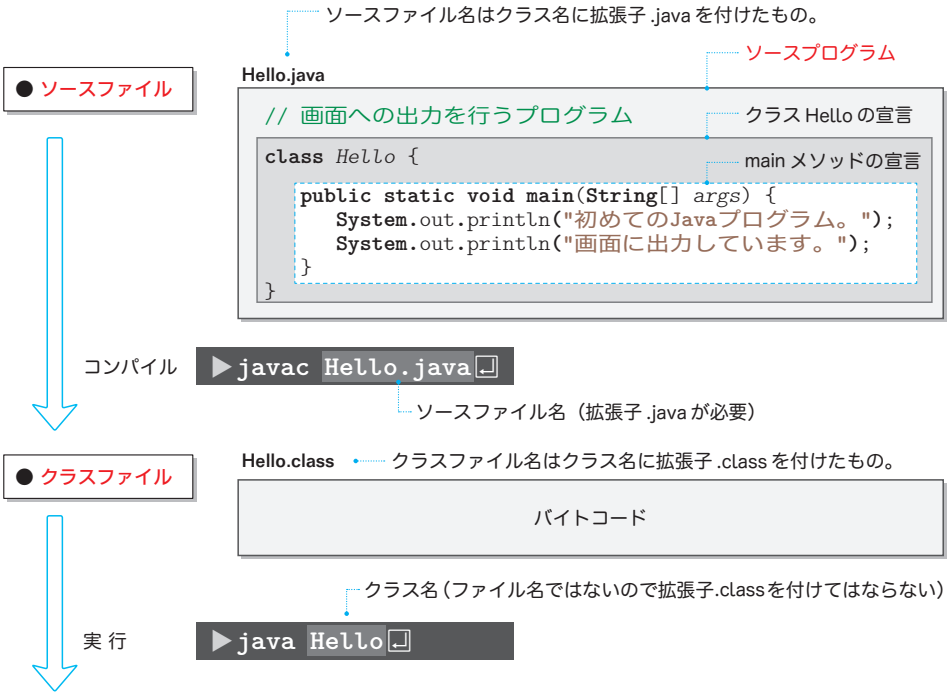

Fig.1-1　プログラムのコンパイルと実行

コメント（注釈）

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、簡潔な言葉で記述するのが、**コメント**（*comment*）すなわち**注釈**（ちゅうしゃく）です。コメントの有無と内容は、プログラムの動作に影響を与えません。記述法には３種類があります。

a 伝統的コメント（*traditional comment*）

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く、**複数行にわたって記述できます。**

```
/*
    a 伝統的コメント
*/
```

b 文書化コメント（*documentation comment*）

注釈を /** と */ で囲みます。a と同様、**複数行にわたって記述できます。**

▶　詳細は第 13 章で学習します。

```
/**
    b 文書化コメント
*/
```

c 行末コメント（*end of line comment*）

// 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。**

```
// c 行末コメント
```

プログラムの構造

本プログラム *Hello* の構造を、**Fig.1-2** を見ながら理解していきます。

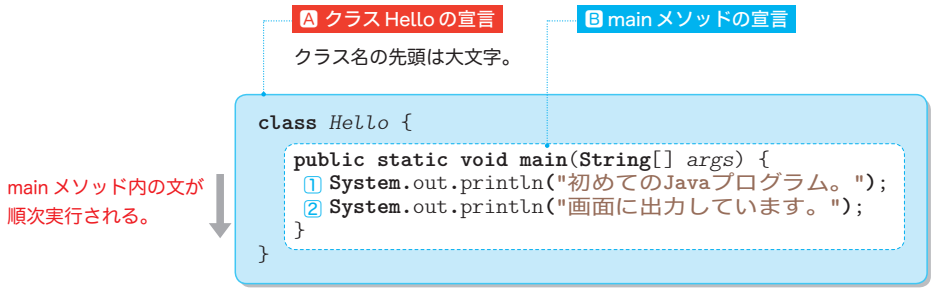

Fig.1-2　プログラムHelloの構造

▪ クラス宣言

Aは、『**名前が *Hello* であるクラス**（*class*）**のクラス宣言**（*class declaration*）』です。

クラス名の先頭文字は、**大文字**とするのが原則です。**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にします。**

▪ main メソッド

Bは **main（メイン）メソッド**（*main method*）の宣言です。詳細は後の章で学習します。

▪ 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、順次実行されます。**文はプログラム実行の単位です。**そのため、まず文1が実行され、それから文2が実行されます。

日本語の文の末尾に句点 。があるのと同様に、Java の文は、**原則としてセミコロン ; で終わります。**

文字列リテラル

"初めてのJavaプログラム。" と "画面に出力しています。" のように、二重引用符 " で囲んだ文字の並びは、**文字列リテラル**（*string literal*）と呼ばれます。

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**）。

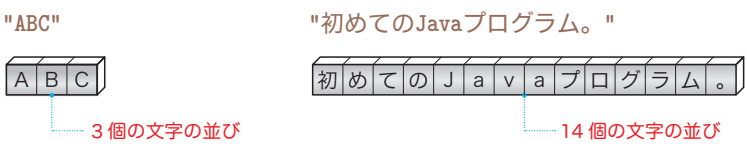

Fig.1-3　文字列リテラル

▶『新・明解Java入門』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。

文とセミコロン

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。そのため、プログラムの実行は不可能です。

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、外部との入出力に利用するのが、“文字が流れる川”にたとえられる**ストリーム**（*stream*）です（**Fig.1-4**）。

`System.out` は、コンソール画面と結び付いている**標準出力ストリーム**（*standard output stream*）です。

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）プログラムの《部品》です。このような部品は、**メソッド**（*method*）と呼ばれます。

前問のプログラム *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

コメント記述時の注意点

伝統的コメントと文書化コメントでは、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します。

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!! */   */
```

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。

▶ すべてのプログラムで同じ実行結果が得られます。

▪ プログラム *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。

▪ プログラム *Hello1B*

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ます。

＊

これ以降、以下のように表現を使い分けます。

「ABC」と表示 … 画面に ABC と表示します。

『ABC』と表示 … 画面に ABC と表示した後に改行します（改行文字を出力します）。

文字列リテラルの連結

複数の文字列リテラルを + で結ぶと、それらの文字列が連結された文字列が生成されます。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。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ましょう。

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

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　バックスラ　バック　※ JIS コードでは ¥
¥	円記号　円　円マーク
$	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
(	左括弧　左丸括弧　左小括弧　左パーレン
)	右括弧　右丸括弧　右小括弧　右パーレン
{	左波括弧　左中括弧　左ブレイス
}	右波括弧　右中括弧　右ブレイス
[	左角括弧　左大括弧　左ブラケット
]	右角括弧　右大括弧　右ブラケット
<	小なり　左向き不等号
>	大なり　右向き不等号
?	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　チルド　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　カレット　キャレット
#	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

注意!!

問題1–4

▶『新・明解Java入門』演習1-2(p.20)

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ましょう。

▪ プログラム *PrintName1A*

４個の文字列リテラル "柴"、"田"、"望"、"洋" を System.out.println メソッドで 1 個ずつ表示します。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。

▪ プログラム *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、以下のように、"洋" の後ろにも \n が必要です。

```
System.out.print("柴\n田\n望\n洋\n");
```

問題1–5

▶『新・明解Java入門』演習1-3(p.20)

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、姓と名のあいだは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

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。

▪ プログラム *PrintName2A*

System.out.println による出力では、**( )** の中を空(から)にできます。以下の文を実行すると、文字が表示されずに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）
```

網かけ部では、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。

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

▪ プログラム *PrintName2B*

姓と名のあいだに、改行文字を表す拡張表記 **\n** 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、姓と名とのあいだが１行あきます。

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。』と試すことがあります。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です。

そこで、よく使われるのが、**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**コメントアウトという手法です。たとえば、*PrintName2A* の網かけ部を

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

とコメントアウトすれば、改行が出力されなくなるため、姓と名があかずに表示されます。

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）で管理するのは、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。

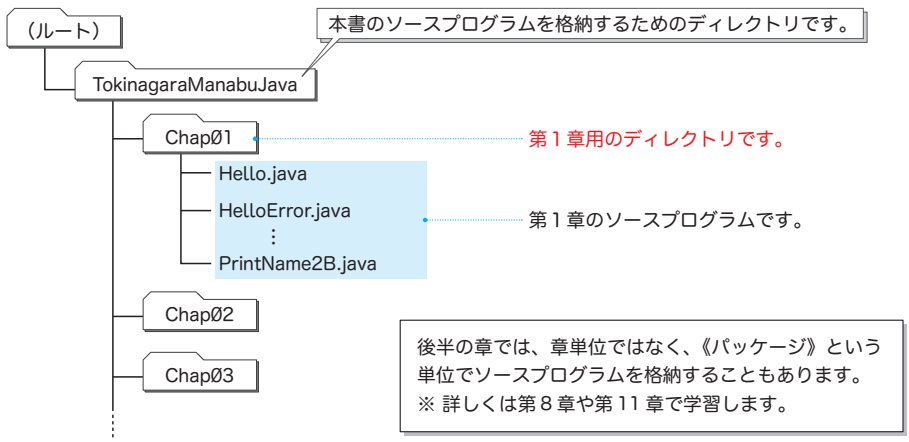

Fig.1-5　本書のソースプログラムのディレクトリ構成（一例）

▶ 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:↵
```

なお、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。

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（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

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 であり、**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れば、基本的に）どこででも動きます。

▪ 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、目的とする処理を容易に行えます。

▪ 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対処を、スマートに行えます。頑丈なプログラムの開発が容易です。