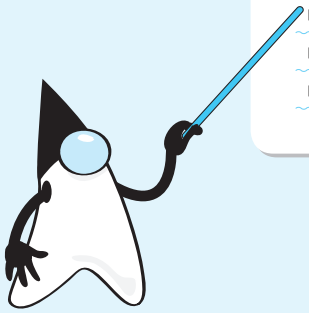
第１章

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。

- □ Java の歴史と特徴
- □ Java プログラムの作成
- □ Java プログラムのコンパイルと実行
- □ コメント（注釈）
- □ 文
- □ 画面への表示とストリーム
- □ 文字列リテラル
- □ 文字列の連結
- □ 改行
- □ インデント
- □ Java 実行環境（JRE）と Java 開発キット（JDK）



1–1 Javaについて

Javaを使ったプログラムの本格的な学習に入る前に、Javaの歴史や特徴などを簡単に学習しておきましょう。

Javaの誕生

1991年頃に、米国のSun Microsystems社が、家電製品用ソフトウェア開発のためのプログラミング言語を作りました。その後に改良が重ねられ、1995年5月のSunWorldで発表されたのが**Java**です。Javaのホームページと、そこから2クリックで行けるAliceのホームページを、**Fig.1-1**に示しています。何だか楽しそうですね。

ちなみに、Javaという名称は、コーヒーに由来した名称であるといわれています。

▶ 初期の開発時の名称はOakでしたが、その名前が他の会社によって登録商標されていたため、Javaという名称に変更されました。

なお、Sun Microsystems社が2010年にOracle社に買収されたのに伴って、Javaに関する権利もOracle社に移っています。

Javaが注目されるきっかけの一つが、Webブラウザ上で動作する《**アプレット**》という小規模なプログラムが開発できることでした。そのため、一時期は『アプレットを作るためのインターネット向けの言語』と誤解された風潮がありました。

Javaは『アプレットを作ることもできる汎用のプログラミング言語』であって、多目的な用途に使える言語です。

▶ 事実、Javaで作られたアプレットはデモ的なものが多く、実用的なものは少ないようです。

http://www.java.com/

http://www.alice.org/

Fig.1-1 Javaのホームページと Aliceのホームページ

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。ここでは、Javaの特徴を簡単に紹介します。

▶ 多少難しい用語などを使っていますので、プログラミングの初心者の方は、いったん読み飛ばして、本書の学習がある程度進んでから読んでいただくとよいでしょう。

▪ 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（**Column 1-3**：p.19）。

▪ 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一種）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るため、オブジェクトの管理が楽です。

▪ 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）は、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 は、C 言語より遥(はる)かに大きく複雑な言語です。本書で学習するのは、Java の基本であって、上記の特徴のすべてを学習するわけではありません（すべてを丁寧に解説しようとすると、数千ページになってしまいますので）。

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 1-1** です。なお、バージョン 5.0 と 6 ～ 8 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 ～ 1.8 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。なお、5.0（内部バージョン 1.5）以降はマイナーバージョンアップはありません。したがって、6.1 とか 7.3 といったバージョンは作られないことになっています。

特に大幅な改訂を受けたのが 1.2 と 5.0（1.5）と 8 の 3 回です。特に、5.0 では、EoD（*Ease of Development*）すなわち**開発容易性**というスローガンが掲げられ、文法体系などが大きく変わり、たくさんの機能が盛り込まれました。

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。

▶ Java には、一般用途向けの Standard Edition (SE) の他に、サーバ向けの Enterprise Edition (EE) や、小型機器向けの Micro Edition（ME）があります。

Table 1-1 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 SE 7	(1.7)	Dolphin	2011 / 7
Java SE 8	(1.8)	—	2014 / 3

学習のための準備

Java を使ってプログラムを開発するためには、

Java 開発キット＝ JDK（*Java Development Kit*）

が必要です。

▶ Java 開発キットは、ソフトウェア開発キット SDK（Software Development Kit）と呼ばれていた時期もありました。

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。

*

なお、以下のホームページで Java に関する解説などを公開していますので、ぜひご覧ください。

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。**本書のすべてのプログラムもダウンロードできます。**

▶ 提供している情報は Java だけではありません。C 言語や C++ 言語、情報処理技術者試験や中国武術など、たくさんの情報を提供しています。

1–2 画面に文字を表示しよう

計算結果を表示しない電卓があるとします。どんなに高速で多機能であっても、おそらく誰も使わないでしょう。文字や数字によって人間に情報を伝えることは、コンピュータにとって重要な仕事の一つです。本節では、コンソール画面に文字を表示する方法を学習します。

プログラムの作成と実行

まず最初に、コンソール画面に文字を表示するプログラムを作りましょう。右に示すように、2行分の表示を行うものとします。

```
初めてのJavaプログラム。
画面に出力しています。
```

テキストエディタなどを使って、**List 1-1** のプログラムを打ち込みましょう。大文字と小文字は区別されますので、ここに示すとおりにします。

List 1-1 Chap01/Hello.java

```
// 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。
```

▶ プログラム中の余白や " などの記号を全角文字で打ち込まないように注意しましょう。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。

本プログラムでは、{ } [ ] ( ) " / . ; と、たくさんの記号が使われています。これらの記号文字の読み方は、**Table 1-2**（p.17）にまとめています。

ソースプログラムとソースファイル

私たち人間は、プログラムを《**文字の並び**》として作成します。このようなプログラムを**ソースプログラム**（*source program*）と呼び、ソースプログラムを格納したファイルのことを**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。ソースプログラムは、《**原始プログラム**》とも呼ばれます。

ソースファイルの名前は、**class** の後ろに書かれている**クラス**（*class*）の名前（このプログラムでは Hello）に拡張子 .java を加えたものとするのが原則です。

したがって、ソースファイルの名前は Hello.java となります。

本書で学習する数多くのソースプログラムのすべて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。そこで、**Fig.1-2** のようにディレクトリとファイルを構成することにします。

お使いのシステムが MS-Windows であれば、ディスク上に MeikaiJava ディレクトリを作り、さらに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。

▶ 本書のプログラムはインターネットのホームページからもダウンロードできます（p.5）。

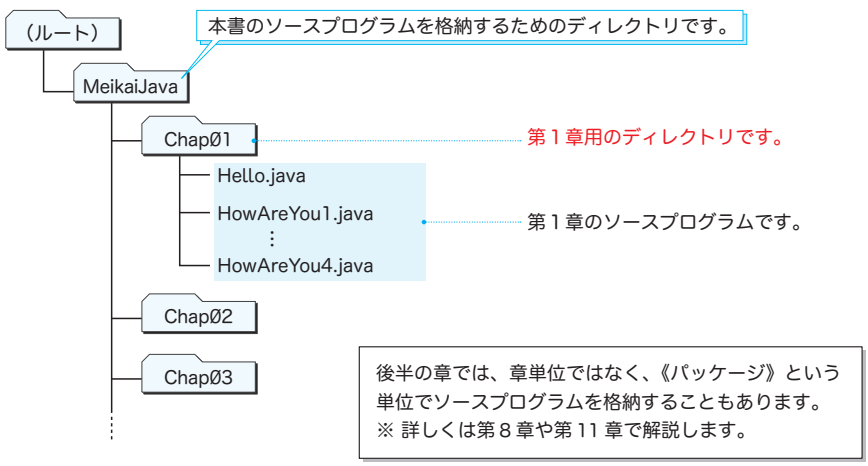


Fig.1-2　本書のソースプログラムのディレクトリ構成（一例）

なお、個々のソースプログラムのディレクトリ名とファイル名は、**Fig.1-3** に示すように、プログラムリストの右上部に示しています。

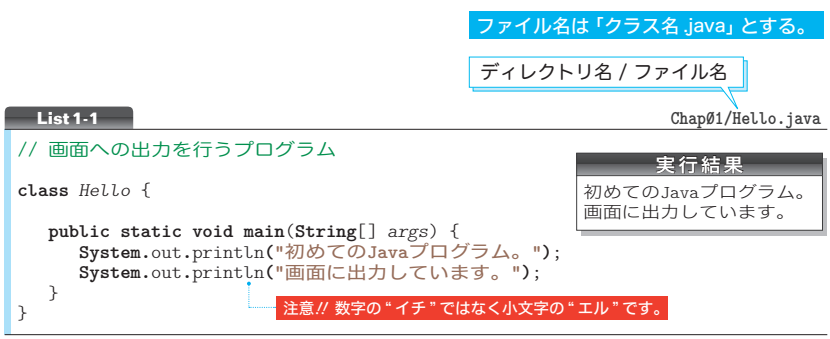


Fig.1-3　ソースプログラムとファイル名

プログラムのコンパイルと実行

ソースプログラムが完成しても、そのままでは実行できません。プログラム実行のためには、**Fig.1-4** に示す二つのステップを踏みます。

a ソースプログラムをコンパイルしてバイトコード（*bytecode*）**を生成する。**
b 生成されたバイトコードを実行する。

Fig.1-4 プログラムの作成から実行までの流れ

a コンパイル

コンパイル（*compile*）とは、そのままでは実行できないソースプログラムを、実行できる形式に変換する作業です。それを行うのが**javac コマンド**です。

Hello.java のコンパイルは、次のようにします（**Column 1-1**）。

```
▶javac Hello.java↵
```

ここで指定するのはソースファイルの名前であり、**拡張子の .java は省略できません。**

▶ タイプするのは白文字の部分です。先頭の▶は、オペレーティングシステムによって表示されるプロンプトですから、タイプは不要です（UNIX では % などと表示されて、MS-Windows では C:\> や C:¥> などと表示されます）。

コンパイルが完了すると、**Fig.1-5** に示すように 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。

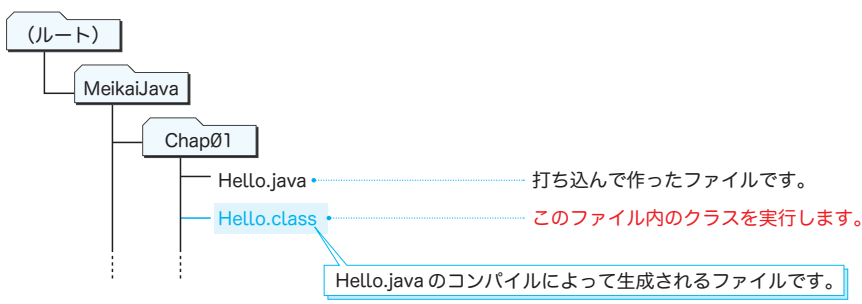


Fig.1-5 ソースファイルとクラスファイル

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。

b 実行

コンパイルに成功したら実行します。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↵
```

コンパイルのときとは異なり、**拡張子 .class を付けてはいけません**（ここで指定するのが、“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枠内に示しています。実行結果は p.6 の **List 1-1** 枠内にあります。

重要 ソースプログラムは、そのままでは実行できない。javac コマンドでコンパイルして、それによって作られるクラスファイル内のクラスを java コマンドで実行する。

Column 1-1 カレントディレクトリ

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。

したがって、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 /MeikaiJava/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 MeikaiJava 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。

コメント（注釈）

プログラムを理解していきましょう。まずは、先頭行に着目します。

```
// 画面への出力を行うプログラム
```

連続する２個のスラッシュ記号 // は、次の指示です。

この行のこれ以降は、プログラムの《読み手》に伝えることがらです。

これは、プログラムそのものというよりも、プログラムに対する**コメント**（*comment*）すなわち**注釈**（ちゅうしゃく）です。

コメントの内容は、**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。

重要 ソースプログラムには、作成者自身を含めた《読み手》に伝えるべきコメントを簡潔に記入しよう。

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。

▶ 本書では、コメントを色付きの文字で表記します。

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い**ため、右のように複数行にわたる際に好都合です。

```
/*
    画面への出力を行うプログラム
*/
```

《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）

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。

```
/**
    画面への出力を行うプログラム
*/
```

▶ この形式のコメントをもとにして、プログラムの仕様書となるドキュメント（文書）の生成が行えるようになっています。第 13 章で学習します。

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はできませんので、手短なコメントの記述に便利です。

```
// 画面への出力を行うプログラム
```

▶ 文書化コメントと伝統的コメントを入れ子にする（コメントの中にコメントを入れる）ことはできません。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!! */
```

Column 1-2 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。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。

プログラムを以下のように書きかえて実行してみましょう。色文字の部分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。");
   }
}
```

実行結果

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（コメント化）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。");
*/
   }
}
```

実行結果

（何も表示されません）

なお、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、などが分からないからです）。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。

プログラムの構造

次に、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。**Fig.1-6** に示す構造です。

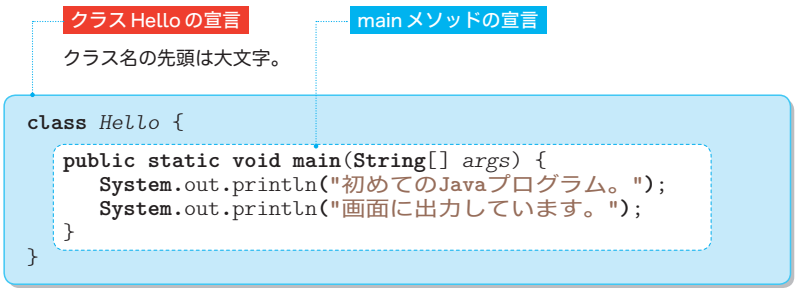


Fig.1-6　プログラムの構造

クラス宣言

図の青い部分は、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、

> *Hello* という名前をもつ**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、以下の形で書くものと覚えておけば十分です。

```
class クラス名 {                    クラス宣言
    // mainメソッドなど
}
```

本プログラムの“クラス名”は *Hello* です。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

図中の白い部分は、**main（メイン）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。

```
public static void main(String[] args) {    main メソッドの宣言
    // 行うべき処理
}
```

文

プログラムを起動して実行すると、**main** メソッドの中の**文**（*statement*）が、順次実行されます（**Fig.1-7**）。

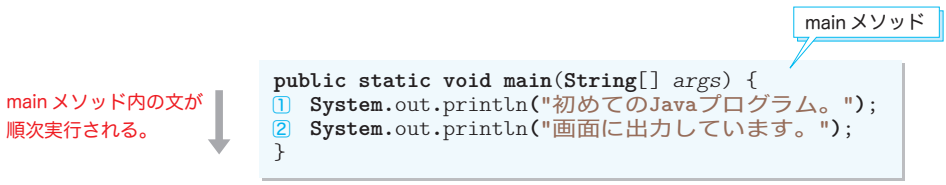


Fig.1-7　プログラムの実行と main メソッド

そのため、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）。

重 要　Java プログラムの本体は **main** メソッドである。プログラム実行時には、その中の文が順次実行される。

▶　メソッドについては、第 7 章以降で詳しく学習します。

文はプログラム実行の単位です。式文・**if** 文・**while** 文 … と数多くの種類がありますので、次章以降で少しずつ学習していきます。

*

さて、**main** メソッド内の二つの文はセミコロン ; で終わっています。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（ただし一部の例外があります）。

重 要　文は、原則としてセミコロンで終わる。

▶　コメントは文ではありませんので、《コメント文》といった文は Java にはありません（そもそもコメントは注釈ですから実行することもできません）。
main メソッドの本体部を囲む { } は、ブロックと呼ばれる一種の文です。ただし、クラス宣言の本体部を囲む { } は、ブロックではありません（そのため、文でもありません）。

演習 1–1

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。

文字列リテラル

コンソール画面への出力を行う文を理解していきましょう。

```
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。");
```

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、そこに書かれているとおりに、3 個の文字 A と B と C の並びを表します（**Fig.1-8**）。

▶ 文字列リテラルの詳細は、第 15 章で学習します。なお、文字列リテラルの他にも、**整数リテラル**・**浮動小数点リテラル**・**文字リテラル**など、数多くのリテラルがあります。
本書では、文字列リテラルを "少し薄い文字" で表記しています。

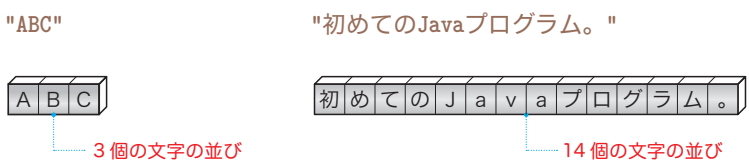


Fig.1-8　文字列リテラル

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（かわ）のようなものです（**Fig.1-9**）。

重 要　外部への入出力は、文字が流れる川である《ストリーム》を経由して行う。

System（システム）.out（アウト）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（プリントエルエヌ）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**改行します**（改行文字を出力します）。

▶ 二重引用符 " は、文字列リテラルの開始と終了を表す記号です。したがって、プログラム実行時に " が表示されることはありません。

本プログラムでは、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。

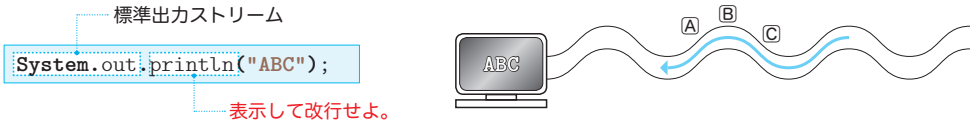


Fig.1-9 コンソール画面への出力とストリーム

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（プリント）を使うと表示後に**改行されません**。**List 1-2** のプログラムで検証しましょう。

▶ HowAreYou1.java というファイル名で保存して、コンパイル・実行します。

List 1-2 Chap01/HowAreYou1.java

```
// 『こんにちは！元気ですか？』と表示

class HowAreYou1 {

   public static void main(String[] args) {
      System.out.print("こんにちは！");
      System.out.println("元気ですか？");
   }
}
```

改行しない

実行結果

```
こんにちは！元気ですか？
```

コンソール画面に『こんにちは！元気ですか？』と表示されます。『こんにちは！』の後で**改行されない**ことが確認できます。

＊

依頼された処理を行う print や println は、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。

▶ プログラムの本体である **main** メソッドも部品の一種です。メソッドの作り方と使い方や **( )** の意味などは、第 7 章で学習します。

本プログラムで利用している二つのメソッドの概要をまとめましょう。

- **System**.out.print(...) … 標準出力ストリームに表示する（改行しない）。
- **System**.out.println(...) … 標準出力ストリームに表示して改行する。

▶ 単語を区切る . については、第 8 章以降で学習します。

println による表示では、**( )** の中を空にすることができます。以下の文を実行すると、文字を表示することなく、**改行だけを行う（改行文字を出力する）**ことになります。

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

なお、これ以降の解説では、以下のように「」と『』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）。

文字列の連結

複数の文字列リテラルを + で結んだら、それらが連結されます。そのことを利用して、前のプログラムを書き直しましょう。それが **List 1-3** のプログラムです。

List 1-3 Chap01/HowAreYou2.java

```
// 『こんにちは！元気ですか？』と表示（文字列リテラルを連結）

class HowAreYou2 {

   public static void main(String[] args) {
      System.out.println("こんにちは！" + "元気ですか？");
   }
}
```

文字列を連結する

実行結果

```
こんにちは！元気ですか？
```

少々わざとらしいプログラム例でした。以下のように、長くて 1 行に入らない文字列リテラルの表示などに + を利用するといいでしょう。

```
System.out.println("むかしむかし、あるところにお爺さんと" +
                   "お婆さんが住んでいました。二人は大の仲良しでした。");
```

改行

文字列リテラルには、《**改行文字**》を表す特別な表記 **\n** を埋め込むことができます。

『こんにちは！』の表示の後に、改行してから『元気ですか？』を表示するプログラムを **List 1-4** に示します。

List 1-4 Chap01/HowAreYou3.java

```
// 『こんにちは！』『元気ですか？』と表示（途中で改行）

class HowAreYou3 {

   public static void main(String[] args) {
      System.out.println("こんにちは！\n元気ですか？");
   }
}
```

改行文字

実行結果

```
こんにちは！
元気ですか？
```

改行を出力すると、それに続く表示は**次の行の先頭**から行われます。そのため、『こんにちは！』が表示された後に、行を改めて『元気ですか？』が表示されます（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。改行文字やタブ文字などのように、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、5–3 節で学習します。

なお、本書では文字列リテラル内の拡張表記を、少し薄い文字ではなく、**真っ黒の太文字**で表記します。

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 1-2** 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、**List 1-4** の表示を行う箇所は、以下のようになります。

```
System.out.println("こんにちは！¥n元気ですか？");
```

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。

Table 1-2　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 !!

自由形式記述

次は、**List 1-5** に示すプログラムを考えましょう。このプログラムは、**List 1-4** (p.16) と実質的には同一であり、実行結果も同じです。

List 1-5 Chap01/HowAreYou4.java

読みにくいけれども正しいプログラム

```
                        /*
 『こんにちは！』『元気ですか？』
と表示（途中で改行）
          */class
HowAreYou4     {
         public static
void main(           String                  [ ]
         args) {
      System     .       out.

println  (
          "こんにちは！\n元気ですか？" )
;}
                           }
```

実行結果

```
こんにちは！
元気ですか？
```

一部のプログラミング言語は、「プログラムの各行を、決められた桁位置から記述せねばならない。」などの記述上の制約を課します。しかし、Java プログラムは、そのような制約は受けません。自由な桁位置にプログラムを記述できる**自由形式**（*free formatted*）が許されているからです。

このプログラムは、思いきり自由に（？）記述した例です。もっとも、いくら自由であるとはいっても、若干の制限があります。

① 単語の途中にホワイトスペースを入れてはならない

class, **public**, **void**, **System**, out, //, /* などは、それぞれが『単語』です。これらの途中に**ホワイトスペース**（スペース文字・タブ文字・改行文字など）を入れて、

```
Sys
   tem
```

と記述することはできません。

② 文字列リテラルの途中で改行してはならない

二重引用符で文字の並びを囲む文字列リテラル " … " も、単語の一種です。そのため、以下のように途中で改行してはいけません。

```
System.out.println("こんにちは！\n
                    元気ですか？");
```

Column 1-3　JRE（Java 実行環境）と JVM（Java 仮想マシン）

JDK（Java 開発キット）は、Java プログラムを開発するツール群に加えて、Java で作られたプログラムを実行するための環境である **JRE**（**Java 実行環境**：Java Runtime Environment）を含んでいます。この包含関係の概略を示したのが **Fig.1C-1** です。

JRE は、**JVM**（**Java 仮想マシン**：Java Virtual Machine）と各種のライブラリなどから構成されています。

コンピュータに JDK をインストールすると、その一部である JRE も同時にインストールされます。ちなみに、『Java のプログラムの開発はしないので、実行さえできればいい。』という場合には、JRE だけをインストールすることができます。

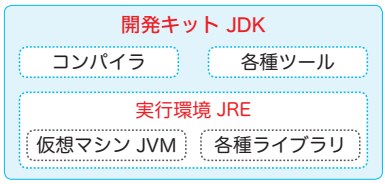


Fig.1C-1　JDK と JRE の関係（概略）

JDK と同様に、JRE は、MS–Windows 用、Mac OS–X 用、Linux 用など、プラットフォームごとに**専用のもの**です（**Fig.1C-2**）。

ただし、javac が生成するバイトコード形式のクラスファイルは、どのプラットフォームの JVM 上でも実行できるものです。このような原理によって、“Write Once, Run Anywhere.” を実現しているのです（p.3）。

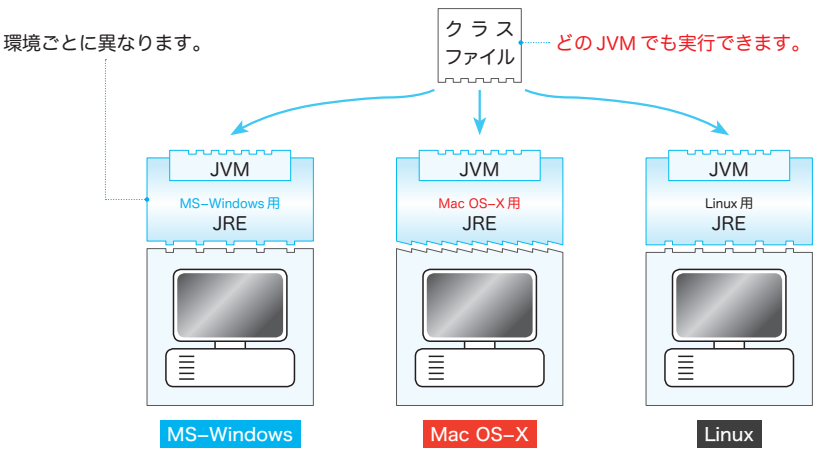


Fig.1C-2　Java プログラムの実行と環境

JVM は、Java プログラムを実行するための仮想マシンであり、クラスファイル中の命令を解釈しながら実行します。しかし、すべての命令を逐一解釈しながら実行しているのでは、十分な実行速度が得られません。そのため、その環境上で高速に実行できるように、クラスファイル中の命令の一部をもう一段階コンパイルします（環境に依存しないように作られたクラスファイルを、現在実行している環境に特化した高速な命令に置換する作業を行います）。

したがって、Java プログラムの実行方式は、逐一解釈しながら実行する**インタプリタ形式**を土台としながらも、機械語を直接実行するコンパイラ形式を利用するという、ハイブリッドな構成となっています。

インデント

もう一度 **List 1-1** から **List 1-4** までのプログラムをよく見てください。**main** メソッドの中に書かれている文は、すべて左端から数えて 7 文字目を先頭に記述されています。

{ }

は、まとまった文をくくる《**段落**》のようなものです。段落中の記述を右に数文字ずらすと、プログラムの構造がつかみやすくなります。

そのために設ける左側の余白のことを**インデント**と呼び、インデントを用いた記述を**インデンテーション**（段付けする／字下げする）と呼びます。

重 要 ソースプログラムにはインデントを与えて、読みやすくしよう。

Fig.1-10 に示すように、本書のプログラムは、左端から 3 文字ごとの幅のインデントを与えています。

▶ すなわち、左側の余白は、階層の深さに応じて 0, 3, 6, 9, … 個分の空白となります。

階層の深さに応じてインデント（段付け／字下げ）します。

```
public static void main(String[] args) {
    for (int i = 1; i <= 9; i++) {
        for (int j = 1; j <= 9; j++) {
            System.out.printf("%3d", i * j);
        }
        System.out.println();
    }
}
```

▶ ここに示しているのは、第 4 章で学習するプログラム（p.134）の一部です。『九九の表』を出力します。

Fig.1-10 インデント

演習 1–2

右に示すように、各行に 1 文字ずつ名前を表示するプログラムを作成せよ。著者の名前ではなく、自分の名前を表示すること。

```
柴
田
望
洋
```

演習 1–3

右に示すように、各行に 1 文字ずつ名前を表示するプログラムを作成せよ。姓と名のあいだは 1 行あけることとし、自分の名前を表示すること。

```
柴
田

望
洋
```

Column 1-4 インデントとタブ文字

インデントについて、幅をどうするのか、タブ文字とスペース文字のどちらを使うのか、といったことを検討していきましょう。

▪ インデントの幅について

インデントの幅は、どのくらいが適当でしょうか。C 言語や C++ では 4 文字分の幅とするプログラムが圧倒的に多いようです。一方、Java では、まれに 8 文字分の幅で書かれたプログラムを見かけるものの、2～4 文字分の幅が主流のようです。

インデントの幅が狭い（2 文字あるいは 3 文字分の幅とする）プログラムが多いのは、以下のような理由によって、Java のプログラムが C 言語や C++ よりも横方向に長くなるからです。

- C 言語の puts や printf が、Java では System.out.print…と非常に長くなること。
- まずクラスがあり、その中にメソッドがあるという構造であるため、インデントの深さが一つ深くなること。

▪ インデントはスペースなのかタブなのか

インデントは、タブキーとスペースキーのいずれでもタイプできます。そのため、多くのエディタは、インデントに関して以下のような機能を提供します。

- タブやスペースをタイプしなくても自動的にインデントを挿入する機能。
- ファイル保存時に、タブとスペースの指定されたほうの文字にインデントを統一する機能。
- タブとスペースの相互変換や置換などの機能。

ここで、注意していただきたいのは、（環境によっては）タイプした文字と、保存したファイル上の文字とが異なる可能性がある点です。タイプ時のインデントと、ファイル上の文字のインデントについて、それぞれの特徴をまとめると以下のようになります。

●タイプ時のインデント

- タブキー　　：タブキーを 1 回タイプするだけですむ。
- スペースキー：スペースキーを何回もタイプしなければならない。

●ファイル上の文字

- タブ文字　　：ファイルが小さくなる。タブ幅の異なる環境ではインデントがくずれる。
- スペース文字：ファイルが大きくなる。環境に依存することなくインデントが保たれる。

環境によってタブ文字の幅は異なります。たとえば、MS–Windows のコマンドプロンプトでは、タブは 8 文字分の幅です（個々のエディタ上では変更できます）。

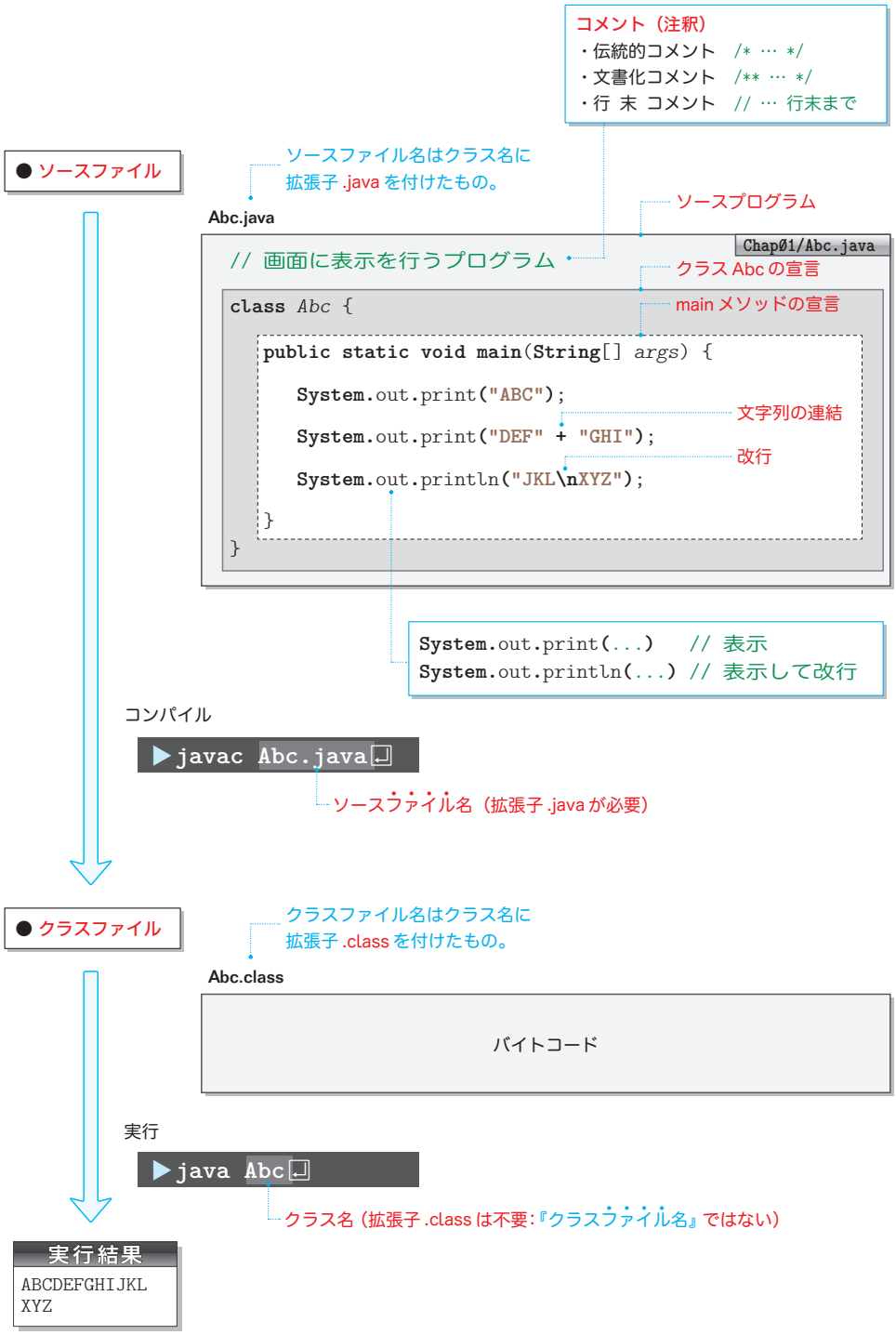
Fig.1-10 のインデントがスペース文字ではなく『タブ文字』であって、それをタブ幅が 8 である環境で表示したのが、**Fig.1C-3** です。間延びしてしまいます。

```
public static void main(String[] args) {
        for (int i = 1; i <= 9; i++) {
                for (int j = 1; j <= 9; j++) {
                        System.out.printf("%3d", i * j);
                }
                System.out.println();
        }
}
```

Fig.1C-3　Fig.1–10 のタブ幅を 8 としたプログラム

まとめ

- Java は、数多くの優れた特徴をもった、**オブジェクト指向プログラミング**をサポートするプログラミング言語である。発表されて以来、改訂が行われるとともに、利用者が増え続けている。
- Java プログラムの開発には、無料で提供される **Java 開発キット**すなわち **JDK** が必要である。
- **ソースプログラム**は《文字の並び》として作成する。それを保存する**ソースファイル**の名前は、**クラス**の名前に拡張子 .java を付けたものとする。
- ソースファイルをそのまま実行することはできない。**javac コマンド**によって**コンパイル**すると、拡張子 .class の**クラスファイル**が作成される。クラスファイルの中身は、**バイトコード**という形式である。
- クラスファイル中のクラスを実行するのが、**java コマンド**である。
- ソースプログラムは、**クラス宣言**の中に **main メソッド**が含まれており、その中に**文**が含まれる構造となっている。
- プログラムを起動すると、**main** メソッド内の**文**が順次実行される。
- 原則として、文の終端はセミコロンである。
- 画面に表示を行う際は**標準出力ストリーム**を利用する。標準出力ストリームに対する文字の出力は、**System**.out.print と **System**.out.println の各**メソッド**によって行える。後者のメソッドでは、出力の最後に自動的に**改行**される。
- 文字の並びを表すのが、二重引用符で文字の並びを囲んだ "…" という形式の**文字列リテラル**である。**+** で結ばれた文字列リテラルは連結される。
- 拡張表記 **\n** は**改行文字**を表す。
- ソースプログラムには、作成者自身を含めた読み手に伝えるべき**コメント**を簡潔に記入すべきである。コメントの記述法には、**伝統的コメント**・**文書化コメント**・**行末コメント**の３種類がある。
- ソースプログラムは、**自由形式**の記述が許される。タブ文字もしくはスペース文字によって、適切な**インデント**を与えて読みやすいものとすべきである。



▶ ほとんどの章の“まとめ”にも、プログラムを示しています。“まとめ”に示すプログラムも理解して実行しましょう。もちろん、ダウンロードできるファイル（p.7）にも入っています。