1. Ｊava 言語とプログラミング

コンピュータやインターネットに興味ある読者なら, Java については耳にしたことがあるだろう。Java は，1995 年に SunMicroSystems 社が発表したプログラミング言語（実際にはプログラミング言語とその言語によって作成されたプログラムを実行するための環境一式）である。コンピュータに関する研究や開発の成果として，現在でも様々な言語が生まれている。その中で, 既存の強力な言語に対抗できるだけの先進性と実用性の両方を兼ね備えた言語だけが, 我々の目に触れて使用できるようになる。
Java はそのような数少ない言語のひとつで，近年では最も注目を集めている。

本草では，以下について学習する。

●Java の特徴
●Java の実行の仕組み
●アプリケーションとアプレット

1．1　Ｊava の特徴

Java は，オブジェクト指向の本格的なプログラミング言語である。オブジェクト指向プログラミング言語は，特に初学者にとっては習得するのが難しいものが多い。
Java の文法は C 言語や C＋＋言語に似ているがずっと整理されており，初学者にとっては比較的学習しやすい言語である。
Java のプログラムは様々なコンピュータ上で実行することができる。これは，Java のプログラムが特定のプラットフォーム（ハードウェアや，OS などのソフトウェア）に依存しないバイトコード（Bytecode）と呼ばれる形態に変換されるからである。Java のプログラムを実行するためには，バイトコードを解釈し，実行するためのプログラムが必要である。このプログラムは仮想マシンと呼ばれ実際の多種多様なプラットフォームに合わせて用意されている。

様々なコンピュータ上で実行されることを考えると，プログラムがちょっとしたことで動作しなくなったり，なんらかの問題の原因になることは避けなければならない。
Java では，C 言語や C＋＋言語では問題の原因となることが多かったメモリ管理について，その大部分を自動的に行うように設計されている。また，コンパイル時と実行時にプログラムをチェックすることで信頼性を高めている。同時に，言語レベルでセキュリティについても配慮している。

インターネット・ブラウザのような大規模なプログラムでは，複数の仕事を同時処理する必要が出てくる。例えば，ユーザとのやり取り，ネットワークでのデータの交換，ウィンドウの表示などを同時に処理する。これらの処理は互いに関連しながらも，比較的独立した処理の流れを持っている。このような処理の流れをスレッド（thread）と呼ぶ。Java では，1 つのプログラム中で複数のスレッドを同時に実行するための機構を備えており，このような機構をマルチスレッド（Multithread）と呼ぶ。プログラムの製作者から見ると，それぞれのスレッドに集中してプログラムを作成でき，プログラムが過度に複雑になるのを防ぎながら，一方でユーザに対しては応答性の良いプログラムを作成することができる。

実用的なプログラムを作成する際には，プログラマが一から十まですべて作成するのではなく，既存の部品をいくつも利用して作成する。このような部品はクラスライブラリ（Class Library）と呼ばれる形で提供される。Java では，このクラスライブラリが非常に充実している。例えば，ファイルなどとの入出力やプログラムで使用頻度の高い部品を集めたクラスライブラリ，アプレットを作成するためのクラスライブラリ，ウィンドウを使用した GUI（GraphicalUserInterface）を作成するためのクラスライブラリ，ネットワークによる通信を行うためのクラスライブラリなど, 高度な機能を実現するクラスライブラリも用意されている。

Java についてもう一点挙げるとすれば，Java がまだまだ成長し続けていることを指摘しておくべきであろう。新しいテクノロジが Java に投入されたり，あるいは Java を用いて実現されたりしている。残念ながら，本書ではそこまで解説することはできないが，Java を通してプログラミングの基礎を学び，同時に Java 言語を習得することは，非常に有益なことであろう。

1．2　Java の実行の仕組み

Java で作成されたプログラムは様々なコンピュータの上で実行することができる。“write at once，run everywhere”（いったん書いたら、どこでも動く）というキャッチフレズが示すように，MS－Windows ベースのコンピュータでプログラムを作成したとしても，Macintosh や UNIX で実行することができる。これまでのプログラミング言語でもこのようなことがまったくできなかったわけではないが，実行速度が遅かったり，どうしても機種ごとの違いが大きかったりして，なかなか実用的に使用するは難しかった。

一般にプログラムを作成する際には，使用する言語の約束や記述方法に従って人間がコンピュータに対する命令を 1 行 1 行に記述していく。ここで作成される（多くの場合，人間が読むことのできる）命令の並びはソースコード（source code）あるいは単にソースと呼ばれファイル（ソースファイル）に保存される。コンピュータはソースコードを直接実行するわけではない。すなわち，何らかの方法でソースコードをコンピュータが実行できる形式に変換しなければならい。この実行できる（executable）形式のプログラムをどのタイミングでどのように作成するかによって，プログラミング言語はコンパイラとインタブリタに大別される。

図 1．1 にコンパイラとインタブリタにおける実行について示す。

コンパイラはソースコードから一括して実行形式を作成する。コンパイラでは、コンパイルと呼ばれる操作でオブジェクトファイルと呼ばれる中間的な形式のファイルを作成し，リンクと呼ばれる操作で複数のオブジェクトファイルを結合（リンク）し，実行形式ファイルで保存する。コンピュータは実行形式ファイルを読み込んで，そこに書いてある命令を実行する。実行形式はコンピュータの機種や OS ごとに異なり，互換性はない。これに対してインタブリタ型では、インタブリタと呼ばれるプログラムが，ソースコードを 1 行ずつ翻訳しては実行する。インタブリタを機種ごとに用意すれば，ソースコードの内容をどこでも実行できるが，1 行 1 行翻訳する作業があるため，実行速度は遅い。

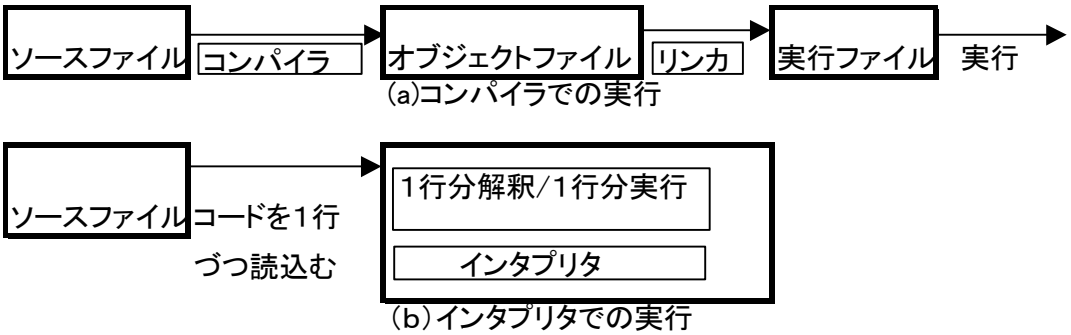


図　1．1　コンパイラとインタプリタ

Java はインタプリタ型である。しかし、Java ではソースコードをコンパイルする必要もある。Java におけるコンパイルは、前述のコンパイラのように実行形式を作るのが目的ではなく、コンピュータの機種や OS に依存しない（Architecture Neutral）形式に変換することが目的である。この形式をバイトコード（Bytecode）と呼ぶ。バイトコードはコンピュータが直接実行

することはできないし，また，他のプログラムの部品とリンクされてもいない。
Java では，各コンピュータごとにバイトコードを実行するプログラムを用意する必要がある。このプログラムは Java 仮想マシン（JVM : Java Virtual Machine）と呼ばれ、現実には存在しない仮想的なコンピュータをソフトウェアで実現したものである。また，バイトコードは、その仮想的なコンピュータの実行形式である。JVM は，バイトコードを実際のコンピュータが理解できる形に逐次変換しながら実行する。すなわち，インタプリタ型の実行を行う。
バイトコードを実行する上でもうひとつ重要なのは，JVM だけではなく，プログラム部品であるクラスライブラリが必要であることである。Java ではクラスライブラリが非常に充実しており，基本的なクラスライブラリについては，プログラムを作成する環境（開発環境）とプログラムを実行する環境（実行環境）の両方で同じものが利用できる。独自のクラスライブラリは，ネットワークなどで伝送することができる。クラスライブラリはプログラムの実行時に JVM に読み込まれその時点でリンクが行われる。

このようにバイトコードを実行するためには，そのコンピュータ上に Java を実行するための道具立て一式が必要である。この道具（といってもプログラムだが）を Java の実行環境（JRE : Java Runtime Environment）と呼ぶ。また，Java 実行環境に開発用のツールを加えたものが Java 開発環境（JDK : Java Development Kit）である。Java の実行環境と開発環境は，Java Soft から無料で配布されている。また，実行環境については，インターネット・ブラウザなどと一緒に配布されている。

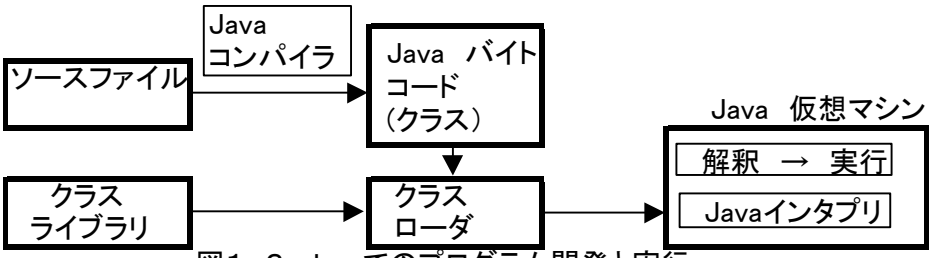


図1．2 Javaでのプログラム開発と実行

1．3 Java プログラムの 2 つの顔ーアプリケーションとアプレット
Java によって作成することのできるプログラムには，アプレット（Applet）とアプリケーション（Application）がある。

アプレットは、インターネット・ブラウザ中で実行することができるプログラムである。ホームページ（HTML ファイル）中で，＜applet＞タグによって実行するプログラムを指定する。ブラウザでそのホームページを表示させると，ブラウザは＜applet＞タグで指定されたプログラムを自動的にダウンロードして実行する。一般に，Java の実行環境はブラウザに含まれる。

アプリケーションは，ワープロやグラフィックス関係のプログラムなど，これまで Java 以外の言語で作成されてきたプログラムと同等と考えてよい。アプリケーションの実行にはブラウザは必要ないが，その代わりに，JVM を含む Java 実行環境が必要である。Java 実行環境は OS にバンドルされていることもあれば，ユーザが自らインストールする必要がある場合もある。

このように，アプリケーションとアプレットでは実行環境が異なる。しかし，プログラムを作成する上ではほとんど同じでものと考えてよく，クラスライブラリについても同じものを使用することができる。ただし，アプレットを実行できることは，ブラウザによって意図的に制

約される。アプレットはディスク上のファイルにアクセスすることができないし，アプレットからの通信は通信相手が制限される。これは，信頼できないホストにアクセスした場合に，アプレットが自動的にダウンロードされて実行されたとしても，そのアプレットが不正なことを行うのを防止するためである。

2．Java でプログラムを作ろう

英語を初めて学んだときのことを思い出してみよう。最初は，主語や述語といった文法より，挨拶や簡単な文例をもとに感覚的に学習しながら，徐々に英語の世界に入っていったはずだ。コンピュータの言語は人工的な言語ではあるが，一通り学習して見通しがつけられるようになるまでは，細かいことや分からないことにとらわれすぎず，どんどん試していくことが習得の早道である。ここでは，Java のプログラムの構造について述べた後，簡単なプログラムを作成して実行するところまでを行う。Java のプログラムにはアプリケーションとアプレットがあるが，プログラムを作成する上で基本となる事項は同じである。　しばらくは，アプリケーションを中心に学習を進める。

本章では，以下について学習する。

●簡単な Java プログラムの構造の概略を理解する
●プログラムの入力と実行までの手順

2．1　簡単なプログラム

Java では，クラス（class）と呼ばれる単位がプログラムを構築する基本的な単位である。クラスは，データや処理を 1 つにまとめたもの，すなわち入れ物のようなものである。クラスの中身（中に入っているデータや処理）をメンバ（member）と呼ぶ。データは一般に変数として表されクラスに属する変数はメンバ変数と呼ばれる。一方，処理を記述したものはメソッド（method）あるいは関数（function）と呼ばれる。

最初の一歩となるプログラムは，“Hello World！”という 1 行の単純なメッセージを画面に表示するプログラムである。プログラムのソースコードをプログラム 2．1 に示す。

```
1:/*はじめてのプログラム*/
2:public class HelloWorld{
3:   public static void main(String argv[]){
4:      System.out.println("Hello World!");
5:      }
6:}
```

プログラム 2.1　Hello World を表示するプログラム

プログラム 2．1 は，全体として HelloWorld という名前のクラスを定義し，そのメンバとして main（）メソッドを定義している。プログラムはたった 6 行だが，それでもこれを正確かつ完全に理解するには若干時間が必要である。以下では，当面必要なことに絞って眺めていくことにする。なお，本書では行頭に「1:…」，「2:…」のように行番号を付けてプログラムを示すが，実際のソースコードでは行番号は不要である。

プログラムの 1 行目のように”/*”と”*/”で囲まれた部分はコメントと呼ばれ適当な文や文字を書くことができる。コメントはプログラムの実行には無関係であり，ソースコードを人間が読む場合に手助けとなる[1)]。”/*”と”*/”でくくる場合は，以下のように，コメントが何行かにわたってもかまわない

```
/*   これはコメントです
        コメントは必ず書くこと   */
```

[1]　Java のソースでは，「/*…*/」，「//…」のほかに「/**..*/」という形式のコメントがよく登場

する。これはドキュメント生成用のツールの JavaDoc で使用される。

このほかに，”//”に続けてコメントを書くこともできる。この場合，//から行末までがコメントになる。4 行目に//を利用してコメントを書く例を以下に示す。

System．out．println（”Hello World”）; // 後で解説します

1 行目はコメントなので, プログラムは実質的に 2 行目の”public class HelloWorld”から始まる。Class はそれに続く名前（ここでは HelloWorld）のクラスを定義することを表している。このプログラムでは 1 つのクラスしかないが，一般には複数のクラスによりプログラムが構成される。先頭の public は, このクラスがプログラムのどこからでもアクセスできる（すなわち public＝公開されている）ことを表している。このように Java では，クラスやメンバに誰がアクセスすることができるかをあらかじめ指定しておくことができる。

2 行目は中括弧開き“{”で終わっている。これに対応する中括弧閉じは，プログラムの最後，6 行目の“}”である。一般に，“｛”から対応する“}”まではブロック（block）と呼ばれ一種のまとまりを表す。ここでは HelloWorld クラスの定義が，プログラムの最後の“}”まで続いていることを表す。

3 行目は“public static void main（String argv［］）｛”である。この行は，HelloWorld クラスのメソッドとして main（）メソッドを定義することを表している。メソッドはいろいろな名前で複数定義できるが，maln（）メソッドは特殊なメソッドである。なぜなら，Java インタブリタは指定されたクラスの main（）メソッドから実行を開始するからである。“static void”や“string argv［］”については，ここでは「このように書くものだ」と考えて深く立ち入らない。しばらくは，main（）メソッドはこの 2 行目のように書くものだと，鵜呑みにしてもらってかまわない。

クラスのときと同様にブロックに着目すると，3 行目末の“{”に対応するのは 5 行目の“}”であることから 5 行目で main（）メソッドの定義が終了していることが分かる。すなわち，main（）メソッドの中身は 4 行目の“System．out．println（“Hello World！”）；”の 1 行である。“println（”HelloWorld！“）”は，“HelloWorld！”を表示せよという意味で，この行全体では System.Out オブジェクトの println（）メソッドの呼び出しを表している。もう少し感覚的に表現すると，8ystem．out オブジェクトに，画面に“HelloWorld！”と表示することをお願いしているといってもよいだろう。

プログラムのソースコードは，空白文字やタブで区切られた単語から成り立っている。注意を要するのは，Java では大文字と小文字は区別される点である。例えば public，Publlc，PUBLIC は異なる単語として識別される。単語には，Java 言語として特定の意味や役割が与えられている単語と，そうでない単語がある。特定の意味や役割が与えられている単語を予約語と呼ぶ。Public，Class などは予約語である。

4 行目のように，文はセミコロン“；”で終了する。Java では（C や C＋＋でも同様だが），改行は文の単位とは無関係である。したがって，1 つの文が複数の行にまたがってもよいし，プログラム 2．1 のソースコードを，改行なしで 1 行にまとめて書いても問題ない。勝手に改行をすると問題が発生する個所もあるが，それ以外では，読みやすいように適当な位置で改行してかまわない。

2．2　プログラムの入力と実行

プログラム 2．1 を実際に動かすまでの手順について述べる。作業の流れを図 2．1 に示す。

Java の開発環境によって具体的な操作は異なるが，プログラムの作成はソースファイルの作成，コンパイル，実行の 3 つの手順で進める。読者は，自分の使用するコンピュータでどのような開発環境が使用できるか確認してみてほしい。MS－Windows 用と Solaris 用の開発環境（Java2 SDK あるいは JDK）は JavaSoft から配布されている。JDK には，Java の実行環境とコマンドラインから使用するコンパイラなどのツールが含まれている。ここでは，MS－Windows 上で JDK を用いてプログラムを作成することを例に解説する。

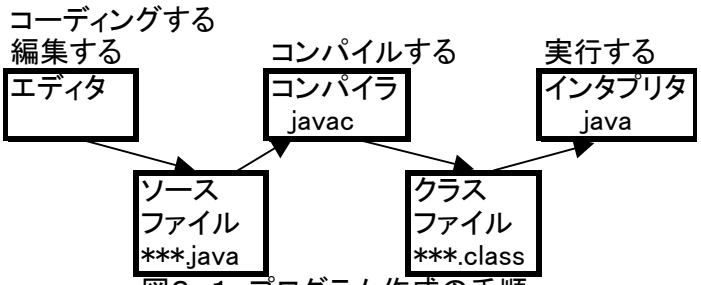


図2．1　プログラム作成の手順

ソースファイルの作成は，「プログラムを編集する」，「コーディングする」などと呼ばれる。ソースコードは，プレイン・テキスト形式のファイルに格納する。一般には，エディタと呼ばれるツールを使用して作成する。ソースファイルには，複数のクラスを格納することができる。ただし，ソースファイルのファイル名はその中に格納するクラスの 1 つの名前に拡張子“. java”を付けたものを使用する。プログラム 2．1 の場合，ソースファイルには単一のクラスが格納され，そのクラス名は HelloWorld なので，ソースファイルの名前は HelloWorld. java である。ここでは，MS－Windows 上でメモ帳を使用して，ソースコードを作成してみる。メモ帳の起動は，スタートメニューから「プログラム」－「アクセサリ」とたどるか，いわゆる DOS 窓のプロンプトで、“notepad”とタイプしてリターンキーを押すと起動できる。メモ帳を用いて，プログラム 2．1 のソースコードを入力する（図 2．2 参照）。ミスタイプや大文字・小文字には気をつけること。

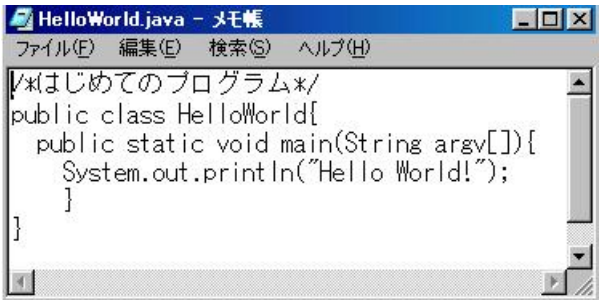
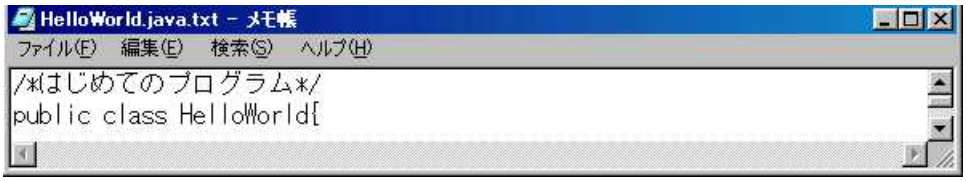


図２．２　メモ帳でのソースコードの入力

ソースコードを入力したら，「ファイル」メニューの「名前を付けて保存…」を選択し，HelloWorld. java という名前で保存する。このとき、どのディレクトリに保存したかを覚えておこう。図 2．3 では、C ドライブの JavaDev ディレクトリ（C：￥JavaDev）に保存している。

保存すると，メモ帳のタイトルバーにファイル名が表示される（図 2．4 参照)。恐らく HelloWorld．java．txt になっているだろう。これでは拡張子が不正（“．java”でない）なので後から修正する必要があるが，ここでメモ帳での作業はいったん終了してよい。

(a) 保存前



(b) 保存後

図２．４　ファイルの保存前と保存後

メモ帳によりファイルを新規作成した場合，ファイル名の拡張子が“．txt”になってしまうので，エクスブローラか DOS 窓からファイル名の拡張子を“．java”に修正する。DOS 窓からは，プロンプト（ここでは C：￥JavaDev＞）に続いて，dir コマンドでファイル名を確認できる。エクスプローラでは，「拡張子を表示しない」設定になっていると“．txt”の部分が表示されず，HelloWorld．java．txt が“HelloWorld．java”と表示されるので注意が必要である。DOS 窓からは，ren コマンドでファイル名の変更ができる。

1：　C：￥JavaDev＞　ren　HelloWorld．java．txt　HelloWorld．java

コンパイルには，DOS 窓で javac　コマンドを使用する。プログラム 2．1 の HelloWorld．java の場合，以下のようにコンパイルする。

２：　C：￥JavaDev＞　javac　HelloWorld．java

コンパイルが正常に終了すると，HelloWorld．java と同じディレクトリ（ここでは C：￥JavaDev）に HelloWorld．class という名前のファイルが作成され画面にはプロンプトが再度表示される。問題があるようなら，コンパイルが正常に終了するまでメモ帳でのソースコードの修正，コンパイルを繰り返す。
クラスファイルができれば，実行することができる。実行は，DOS 窓で java コマンドを使用する。HelloWorld クラスは，以下のようにして実行する。

3：C：￥JavaDev＞java HelloWorld

“．class”が不要であることに注意してもらいたい。これは，HelloWorld クラスを実行することを指定しているからである。java コマンドは，HelloWorld クラスをファイル HelloWorld．class から読み込む。実行結果として，“HelloWorld！”と画面に表示されるはずである（図 2.

5 参照)。

図２．５　ソースファイル名の変更、コンパイル、実行

3. データ型

一般に「情報が豊富だ」とか「データがない」という言葉を使うが，この場合の情報やデータは，その内容を文章や数，映像などで表現したものである。内容は同じでも，情報には表現の形がいくつも考えられる。プログラミングの用語では，情報の表現形式を型（type），あるいはデータ型と呼ぶ。プログラムの中で取り扱うデータは，厳密にどのような表現かが決められている。情報を取り扱う最も基本的な形は変数である。

本章では，以下について学習する。

●コンピュータでの情報の取り扱い
●変数の宣言と使い方
●基本となるデータ型
●参照型と配列
●文字列と画面への出力

3．1　情報量とデータ長

長さを測る単位としてメートル，重さ（質量）を量る単位としてキログラムがあるように，情報量を量る単位としてビット（bit）がある，ビットは情報量の最小単位であり，1 ビットで 0 か 1，2 つの状態のどちらであるかを表すことができる。例えば雨を 0 で表し，晴れを 1 で表すことにすれば，「今日の天気は 0 である。」，または「今日の天気は 1 である。」というように情報が伝えられる。もう少し細かく天気を表現したければ，2 ビット使って，雨（00），雪（01），曇り（10），晴れ（11）のように，4 つの状態を表現することができる。

これを拡張していくと，3 ビットで 8 種類，4 ビットで 16 種類，そして 8 ビットは 00000000 から 11111111 の 256 種類の状態が表現できる。この 8 ビットを 1 つのまとまりとして 1 バイト（byte）と呼ぶ。コンピュータでは，64 メガバイトのメモリとか 1 ギガバイトのディスクというように，通常，基本単位としてバイトを使用する。データを格納するのに必要な長さはデータ長と呼ばれ Java で使用されるデータは，1，2，4，8 バイトのいずれかの長さを基本とする。データをメモリに格納する際には，最低でもそのデータのデータ長と同じ大きさのメモリ領域を必要とする。

演習 3．1
40 人のクラスの出席番号を付けるには，何ビット必要か。

3．2　コンピュータでの数値表現

コンピュータでは，すべての情報は 0 と 1 の組み合わせで表されている。それでは，われわれが日常使っている数値はどのように表現されるのだろうか。例えば，1 つの数値を表すのに 4 ビット使用，すなわち，データ長が 4 ビットだと仮定すると，われわれが使用している 10 進数とコンピュータ内部の表現である 2 進数の間で，図 3．1 のような対応付けが考えられる。このような数値の表現法を 2 の補数表示という。

0000	0		
0001	1	1111	−1
0010	2	1110	−2
0011	3	1101	−3
0100	4	1100	−4
0101	5	1011	−5
0110	6	1010	−6
0111	7	1001	−7
		1000	−8

図 3．1　2 の補数表示を用いた数値表現

すなわち，2 の補数表示では，データ長が 4 ビットの場合には，−8 から＋7 までの数値を表現することができる。同様に，データ長が 1 バイトのときは，−128 から＋127 までを表現することができるる。また，先頭の 1 ビットは符号を示し，このビットを符号ビットと呼ぶ。

3．3　変　数

プログラム中で，データ（値）を保持するために変数を使用する。例えば，ある銘柄の株価や入力されたあなたの名前を保持したり，あるいは平均点や偏差値のように，プログラムで計算した結果を保持したりするには，変数を使用する。

以下では，変数を使用する前に必要な宣言と，プログラム中で変数をどのように使用するのかについて解説する。

3．3．1　変数と宣言

変数は，使用する前に宣言を行わなければならない。変数の宣言は，最初に型（type）を，それに続けて変数名を書く。数学などで「変数 x」や「変数 y」といった呼び方をした記憶があるだろう。この“x”や“y”は変数名に相当し，この名前で変数を区別する。型は，その変数がどのような情報を表すのか（例えば，数なのか文字なのか）を示す。変数の宣言はブロック中のどこに書いてもよい。変数の宣言はセミコロン（；）で終了する。

以下に，整数を表す変数の変数宣言の例を示す。整数を表す代表的な型は int 型である。この例では，2 つの整数型の変数 x と y を宣言している。

```
//  整数を表す変数 x と， x とは別の整数を表す変数 y の宣言
int x ;
int y ;
```

この例のように，変数の宣言を何行かにわたって分けて書いてもよい。また，型が同じ変数を複数宣言する場合は，以下の i と j のように，変数名をカンマ（,）で区切って並べて書くこともできる。

```
//  整数を表す変数 i と j の 2 つを同時に宣言
int  i, j ;
```

変数名は，いくつかのルールを守れば自由に決めてよい。次のことに注意して分かりやすい

名前を使用するべきである。

●変数名は英字（英大文字，英小文字，アンダーバー“＿”，ドル“＄”など[1]）で始まり，いくつかの英字か数字を並べたものである
●名前の途中で空白（スペースやタブなど）を入れることはできない
●大文字と小文字は区別される
●Java で使用方法が決まっている予約語は変数名として使用できない

変数名は x や y のように 1 文字でもよいが，英単語やローマ字義記の日本語の単語（あるいはそれらを並べたもの）などもよく使用される。

3．3．2　値の代入と初期化

変数に値を設定する操作を代入と呼ぶ。宣言済みの変数に値を代入するには，イコール（“＝’’）を使用して，以下のように記述する。

```
x ＝10；　//　変数 x に値 10 を代入，変数 x はこの行以前に宣言されている
```

上記の“x ＝10；”は、数学なら「x の値は 10 である」を意味するが，Java では「変数 x に値 10 を代入せよ」という意味になる。

変数を宣言した後に，変数に初期値を設定する操作を変数の初期化[2]と呼ぶ。Java では，変数を宣言する際に変数名に続いて値を示すことで，宣言と初期化を同時に行うことができる。

```
int x＝0　；　//　変数 x を宣言して，0 に初期化する
```

上記のような初期化を行わない場合は，変数を宣言した後に適切な値を代入する。

```
int x；　//　変数 x の宣言
x＝0；　//　0 を代入
```

初期化も代入も行わず，したがって値が設定されていない変数を参照するようなプログラムを書くと，コンパイル時にエラーとして検出される。

[1] 変数名には，これらの文字のはかにアルファベットを表すユニコード文字が使用できる。
[2] 一般に，変数を宣言した後ではじめて値を設定する操作は初期化と総称されるが，文法としては，初期化と代入は区別される。

3．4　Java における基本データ型

Java では，基本となる型として 8 種類の基本データ型が提供される。図 3．2 に基本データ型の種類について示す。基本データ型は，数を表す型と真か偽を表す論理型に大きく分類できる。数を表す型は，さらに整数を表す整数型と実数（つまり小数）を表す浮動小数点型に大別される。

基本データ型の変数を宣言すると，プログラムの実行時には，その型に応じてメモリ上の記憶領域が自動的に確保される。プログラム中では，変数名を通して，その記憶領域に値を格納したり，格納した値を使用することができる。

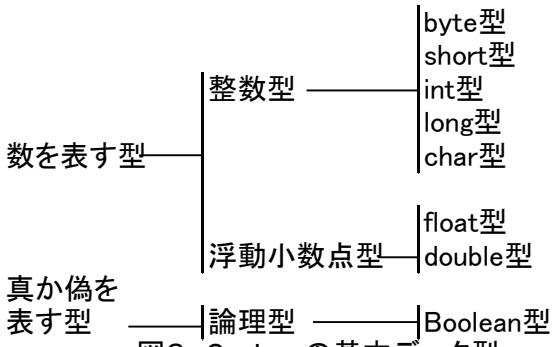


図3．2　Javaの基本データ型

3．4．1 整数型

正または負の整数を表現するには，整数型を使用する。整数型には，データ長によって byte 型，short 型，int 型，long 型がある。 c har 型も整数型の 1 つであるが，使い方が他と異なるので別途解説するものとし，ここでは上記の 4 つの型について述べる。

整数型のデータ長と表現できる値の範囲を表 3．1 に示す。いずれも符号付きで，2 の補数表示により整数を表現するが，データ長が異なる。最もよく使用されるのは，データ長が 4 バイトの int 型で，それより広い範囲の整数を取り扱うときはデータ長が 8 バイトの long 型を使用する。

Java で書かれたプログラムはいろいろなコンピュータで実行される可能性があるが，データ型のデータ長は常に同じものが使用される。例えば，int 型は MS－Windows でも Macintosh でも UNIX でも 32 ビット（4 バイト）整数である。

表3．1　Javaの整数型

型	データ長	範囲
byte	1バイト	-127から128
short	2バイト	－32，768から32，769
int	4バイト	－2，147，483，648から2，147，483，647
long	8バイト	－9，223，372，036，854，775，808から9，223，372，036，854，775，808

一般に，整数を書き表すのに 10 進数（decimal）が用いられるが，プログラム中では，10 進数のはかに 16 進数（hexadecimal）や 8 進数（octal）を使用することができる。10 進数の具体的な値をプログラム中で書く場合は，位取りのためのカンマ（,）などを入れずに以下のように書く。

10 進数の値の例：　12345678

特に値が long 型の場合，最後に“L”または“l”（L の小文字）を付ける。

long 型の値の例：　12345678L または 123456781

“0x”で始まる数値は 16 進数を表し，“0”で始まる数値は 8 進数を表す。

16 進数の例：　0xBC614E または 0xbc614e
8 進数の例：　057060516

プログラム 3．1 に、int 型と long 型の変数を宣言して，適当な整数で初期化し，表示するプログラムの例を示す。このプログラムでは，最初に int 型の変数 x を宣言し（3 行目），別の文（4 行目）で初期化している。long 型の変数 y については，宣言と初期化を同時に行っている（6 行目）。ここで，変数 y の宣言は x への値の代入の後に行われているが，Java では，変数が必要になった時点で宣言して使用することができる。

変数の値の表示には System．out．println（）を使用している（8，9 行目）。その使用方法は 3．6 節で述べる。ここでは，最後の括弧の中に記述した変数の値が画面に表示されると理解していただきたい。

```
1:public class TestIntVal {
2:   public static void main(String argv[]){
3:    int x;                  //int 型の変数 x の宣言
4:    x = 10;                 //x への値 10 の代入
5:
6:    long y = 20L;           //long 型の変数 y の宣言と初期化
7:
8:    System.out.println(x);      //値の表示
9:    System.out.println(y);
10:    }
11:}
```

プログラム３．１　整数型の変数の使用例（TestIntVal.java）

演習 3．2

プログラム 3．1 を実際に作成し，動作を確認しなさい。long 型の変数に int 型の値を代入したり，int 型の変数に long 型の値を代入すると，どのようなことが起きるか。

3．4．2　浮動小数点型

正または負の実数を表現するためには浮動小数点型を用いる。整数と同様，データ長によって float 型と double 型がある。コンピュータの内部では 2 進数で演算が行われるため，10 進数で表現されている数値はいったん 2 進数に変換される。整数の場合は問題ないが，実数の場合には，2 進数に変換されると若干の誤差が生じ，近似値で表現される。ちょうど，分数（例えば 1/3）を小数で表現したときに，近似値で表現されるのとよく似ている。この変換による誤差を丸め誤差という。

浮動小数点型のデータ長と表現できる値の範囲を表 3．2 に示す。浮動小数点型は，内部的には 2 のべき乗の形で保持される。すなわち，実数を仮数部×$2^{指数}$の形で表現し，float 型なら仮数部を 23 ビット　指数部 8 ビット, 符号を 1 ビットの計 32 ビット（4 バイト）で表現する。double 型の場合は，仮数部を 52 ビット，指数部 11 ビット，符号を 1 ビットの計 64 ビット（8 バイト）で表現する 3。float 型の有効桁数は 6 から 7 桁であり，double 型の有効桁数は 15 から 16 桁である。double 型は float 型の約 2 倍の精度を持つことから, 倍精度（double precision）型などとも呼ばれる。

浮動小数点型では，精度を優先して double 型が使用されることが多い。

表3．2　Javaの浮動小点数型

型	データ長	範囲
float	4バイト	約±3.40282347e±38の範囲
double	8バイト	約±1.79769313486231570e±308の範囲

プログラム中で 3．14 のように書いた場合，double 型の数値と解釈される。明示的に double 型であることを示したい場合は，末尾に“D”または“d”を付ける。float 型の場合は，末尾に“F”または“f”を付ける。整数部と小数部のどちらかは省略することができる。整数部を省略する場合はドット“．”で始め，例えば 0．34 は．34 のように書く。指数部を書く場合は，振数部に“E”または“e”を加え，それに続けて指数部を書く。

double 型の例：3．14　　3．14D　　3．14d　　3．14E＋10d
float 型の例：　3．14F　　3．14f　　3．14E－2f

プログラム 3．2 に，double 型を例に浮動小数点型の変数の使用例を示す。このプログラムでは，2 つの変数 g と c を同時に宣言し（3 行目），それぞれに double 型の値を代入し（4，5 行目），表示する（7，8 行目）。

```
1:public class TestDouble{
2: public static void main (String argv[]){
3:   double g, c;
4:   g = 9.80665;              //重力加速度(m/(s*s))
5:   c = 2.99792458e+8d;   //光の速度(m/s)
6:
7:   System.out.println(g);   //値の表示
8:   System.out.println(c);
9: }
10:}
```

プログラム３．２　浮動小数点の変数の使用例（TestDouble.java）

演習 3．3
　float 型と　double　型の変数をそれぞれ定義し，適当な値を代入した後，System．out．println()により表示しなさい。

3．4．3　文字型

文字を表現するためには char 型を使用する。例えば，int 型で 1 つの整数を表現できるのと同様，char 型では 1 つの文字を表現できる。単語や文章は文字が並んだものであるが，これらは文字の列（文字列）であり，文字とは区別される。

プログラム中では，文字はシングルクオート“‘”でくくって表す。例えば，文字 a は’a‘，文字 7 は’7‘のように書く。プログラム中で，3 と書くか’3‘と書くかで，数値か文字かの違いが生じることに注意を要する。

char 型は分類としては整数型であり，データ長は 2 バイトである。コンピュータの内部では番号（整数）で文字を表しており，この番号を文字コードと呼ぶ。英文で使用するような文字，すなわち，英字，数字，主だった記号は 256 文字以下で，1 バイトで表現することができる。しかし，ひらがなや漢字を含む日本語の文字を表現するには，2 バイト必要である。Java では，

日本語を含めた様々な言語の文字を使用できるように 2 バイトの char 型が採用されている。

文字コードの中でもっとも広く用いられているのは，ASCII（American Standard Code for Information Interchange）コードである。このコードは英字，数字，記号を 0 から 127 の値で表すが，ひらがなや漢字は含まれない。このコードでは，例えば文字‘a’には 10 進数で 97，16 進数で 0x61 のコードが与えられている。一方，Java ではユニコード（unicode）が用いられる。ユニコードはまだ一般に馴染みが薄いコードではあるが，2 バイトで 65，536 文字を表現することができる。実は，ユニコードの最初の 128 文字に ASCII コードが割り当てられている。ユニコードは，先頭に“￥u”を付けた 16 進数で表現することが多い。例えば，‘a’は‘￥u0061‘である。

通常は文字と考えられていないものでも，コンピュータの内部では文字として扱うものがある。例えば，改行も Java では 1 つの文字として扱われる。このような特殊な文字は，複数の文字を使用して表現し，これをエスケープシーケンス（escape sequence）と呼ぶ。主なエスケープシーケンスを表 3．3 に示す。

表3．3　主なエスケープシーケンス

エスイケープシークエンス	意味	コード
¥b	バックスペース	¥u0008
¥t	タブ	¥u0009
¥n	改行	¥u000a
¥f	フォームフィード	¥u000c
¥r	キャリッジリターン	¥u000d
¥″	ダブルクオーテーション	¥u0022
¥'	シングルクオーテーション	¥u0027
¥¥	¥記号	¥u005c
¥(8進表記)	0から0377までの文字コードを直接記述	
¥u(16進表記)	文字をユニコードで直接記述	

プログラム 3．3 に，文字型の変数の使用例を示す。このプログラムでは，3 つの文字型の変数 cl，c2，c3 を宣言している（4 行目），いずれも文字‘a’を代入している。

“cl＝‘a’;”（4 行目）は直感的に理解できるだろう。変数 c2 には，‘a’を文字コード（16 進数）で表現して代入している（5 行目）。変数 c3 には，‘a’をユニコードで表現して代入している（6 行目）。ユニコードがシングルクオート“‘”でくくられていることは注意が必要である。

```
1:public class TestChar{
2: public static void main (String argv[]){
3:   char c1, c2, c3;
4:   c1 = 'a';            //文字 a
5:   c2 = 0x61;          //やはり文字 a、文字コードは 16 進数で 61
6:   c3 = '¥u0061';  //これもやはり文字 a、ユニコードで 0061
7:
8:   System.out.println(c1);  //値の表示
9:   System.out.println(c2);
10:   System.out.println(c3);
11:}
```

12:}
プログラム３．３　文字列の変数の使用例（TestChar.java）

演習 3．4
　以下のように，整数型の変数に文字を代入して値を表示すると，文字コードが表示できる。あなたのイニシャルを表す英文字の文字コードを表示するプログラムを作成しなさい。また，あなたの氏名の最初の文字（漢字やひらがな）についても調べなさい。

```
int　myInitialCode1 = 'K',　myInitialCode2='岡';
System.out.println(myInitialCode1);
System.out.println(myInitialCode2);
```

3．4．4　論理型
　真か偽を表現するには boolean 型を使用する。プログラム中で，真は true，偽は fa1se と書く。例えば，ある条件が成立する場合には true，成立しない場合は fa1se のように，状態を表すのに頻繁に使用される。3．1 節でビットについて説明したが，boolean 型は 1 バイトで真か偽を表現し，数値を表現する型とは区別されている[4]。

　Boolean 型の例：　true　　fa1se

[4]　C 言語では論理型はなく，整数型で代用されている。C＋＋言語では bool 型が定義されており，整数との間で変換できる。Java では，boolean 型は整数とはまったく別のものして扱われキャストを使用しても相互に変換できない。

3．5　参照型と配列
Java では，基本データ型のほかに参照型と呼ばれる変数がある。参照型を用いるデータ型として，配列や後で出てくるクラス型などがある。int　x；のような基本データ型の宣言では，変数 x に対応するメモリ上の記憶領域は自動的に確保される。しかし，参照型では，記憶領域は自動的に確保されず，プログラムで明示的に記憶領域を確保する必要がある。また，図 3．3 に示すように，基本型の変数は直接的に値が格納されている記憶領域を表しているのに対して，参照型の変数は値が格納されている記憶領域の位置が格納される。この位置のことを参照と呼ぶ。

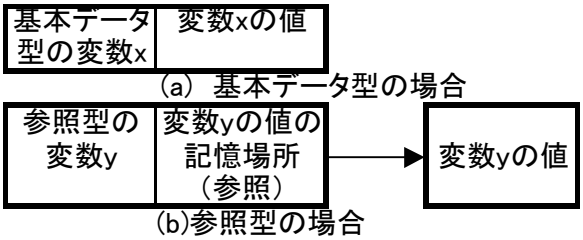


図3．3　基本データ型と参照型

配列は，例えばベクトルや行列のように，同じ型の変数の並びである。配列を表す変数の宣言は以下の通りである。

型[ ] 変数名 ； または 型 変数名[ ] ;

これは基本データ型の宣言と非常に似ているが，“[ ]”が付いている点に注意が必要である。要素が int 型の値で，それを複数並べるような配列の例として，3 次元のベクトル vec を考えた場合，vec は配列で表すことができ，その宣言は以下の通りである。

int[ ] vec ； // int 型の配列 vec の定義

この宣言では，要素の数がいくつであるかは指定せず，あくまで，vec は int 型の配列であることが宣言される。より正確には，int 型の配列への参照として変数 vec を宣言している。宣言した時点では，具体的な位置については不明，すなわち，どこも参照していない。このような状態はプログラム中では予約語 null で表され vec の初期値は null である。

配列の参照型の場合は，変数の宣言とは別に実際の記憶領域を確保する必要がある[5]。すなわち，配列を生成する必要があり，これには演算子 new を使用する。生成した配列は，以下のように変数に割り当てる。

```
int［］vec；            // int 型の配列 vec の宣言
  vec＝new int［3］；  // 配列の生成
```

または,

```
int［］vec＝new int［3］； // int 型の配列 vec の宣言と配列の生成
```

この例では，3 次元のベクトルを表すための配列を生成するので要素数は 3 である。配列の生成の際に，この要素数 3 を指定している。

配列の要素を数える際には 0 から数える。すなわち，要素数 3 の配列とは，要素 0 から要素 2 の合計 3 個の要素から構成されることになる。配列 vec の各要素は，プログラム中では以下のように表現される。

```
vec［0］；     // 配列 vec の 1 番目（先頭）の要素
vec［1］；     // 配列 vec の 2 番目の要素
vec［2］；     // 配列 vec の 3 番目（最後）の要素
```

例えば，配列 vec の 2 番目の要素に値 5 を代入するには以下のようにする。

```
vec［1］＝5；     // 配列 vec の 2 番目の要素に値 5 を代入
```

[5] ここでは記憶領域の確保と表現しているが，正確には後で解説する，オブジェクト（インスタンス）の生成を指す。

プログラム 3．4 に，配列を使用した例を示す。このプログラムは，3 次元のベクトルを表す

配列 vec を宣言し（3 行目），ベクトル（1，2，3）になるように各要素に値を代入している（6～8 行目）。その後，1 番目（先頭）の要素と 3 番目（最後）の要素の値を入れ替えて（11～14 行目），ベクトル（3，2，1）を作成し，各要素の値を表示している（17～19 行目）。

```
1:public class TestArray{
2: public static void main (String argv[]){
3:   int[] vec = new int[3];     //配列 vec の宣言と配列の生成
4:
5:/*   各要素に値を代入   */
6:   vec[0] = 1;
7:   vec[1] = 2;
8:   vec[2] = 3;
9:
10:/*  先頭と最後の要素の値を入れ替え   */
11:   int tmp;
12:   tmp = vec[0];
13:   vec[0] = vec[2];
14:   vec[2] = tmp;
15:
16:/*  各要素の値の表示   */
17:   System.out.println(vec[0]);
18:   System.out.println(vec[1]);
19:   System.out.println(vec[2]);
20: }
21:}
```

プログラム３．４　配列の使用例（TestArray.java）

3．6　文字列と画面への出力

文字の並び，すなわち文字列は，基本データ型の文字（char）型とは明確に区別される。文字列は，Java のプログラム中ではダブルクオート（“）でくくって表現される。

```
“abc”;   //  文字列   “abc”
```

文字列を扱うためには，後で述べるクラスを使用する必要がある。ここでは詳細には立ち入らず，画面への出力に使用する程度のことを表記的に説明する。

文字列は，“＋”を用いて連結することができる。文字列“abc”と“def”を連結したものは文字列“abcdef”であり，“abc”＋“def”と表現することもできる。既にサンプルプログラム中で使用しているが，System．out．println（）は文字列を画面に出力する。例えば，文字列“abc”を画面に出力するには，以下のようにする。

```
System．out．println（“abc”）;   //  文字列   “abc”の画面への出力
```

ここで，文字列“abc”と“def”を連結した結果を表示するには，以下のようにする。

```
System．out．println（“abc”＋“def”）;   //  文字列   “abcdef”の画面への出力
```

出力の際には，特に基本データ型の変数はその値が自動的に文字列に変換される。したがって，変数 x の値を出力するのに，System．out．println（ｘ）; のようにすればよかったわけである。ここで，先はどの文字列の連結を利用すると，以下のように，利用者に理解しやすい形で変数の値を出力することができる。

System．out．println（“x＝”＋x）;

上記のプログラムで，変数 x の値が 10 なら，その出力は以下の通りである。

x　＝10

演習 3．5
2 つの 3 次ベクトルを int 型の配列により実現し, 2 つのベクトルの和を表示するプログラムを作成しなさい。ベクトル a と b の 2 番目の要素の和は，以下の通りである。
a［1］＋b［1］

4．演算子

コンピュータに計算させるという場合に，多くの読者がイメージするのは数値の加減乗除だろう。コンピュータの用語では，計算とともに演算という言葉が使用されるが，実際には，値の大小関係を判定したり，ビット単位でデータを操作したり，あるいは，既に紹介した変数への値の代入といったことも演算に含まれる。

本章では，以下について学習する。

●算術演算子
●代入演算子
●インクリメント演算子とデクリメント演算子
●関係演算子と論理演算子
●ビット演算子
●混合演算とキャスト

4．1　算術演算子

いわゆる四則演算を行うための算術演算子には，以下の 5 つがある。

```
加算：＋        a＝5＋2；   ⇒7
減算：－        s＝5－2；   ⇒3
乗算：＊        t＝ 5＊2；  ⇒10
除算：／        d＝5／2；   ⇒2
剰余算：％      m＝5％2；   ⇒1
```

加算と減算の演算子は数学の表記と同じ“＋”と“―”であるが，乗算は“×”の代わりに“＊”を，除算は“÷”の代わりに“／”を使用する。剰余算は余りを求める演算[1]で，整数型データ間に限られる。それ以外は，すべての数値データ型に使用することができる。

ただし，除算に関して，整数型データ間の除算では小数点以下が切り捨てられるので，浮動小数点を含む除算とは結果が異なることがある。

```
浮動小数点を含む除算：5．0／2．0；      ⇒  2．5
整数型データ間の除算：5／2；            ⇒  2
```

[1]　a%b = a-(a/b)*b で定義される。

同じ式に複数の演算子が含まれる場合，どの演算から行うかについては決められている。算術演算子の場合，演算子が出現した順番だけでなく，数学と同じように“＊”や“／”が優先して計算される。これを意識的に変更する場合は，これも数学での表現と同様に，優先する演算を括弧“(”,”)”でくくる。

```
//　上底 a，下底 b，高さ h の台形の面積 s を求める。
  s＝a＋b＊h／2．0；        // 誤り。s は b＊h／2．0　と a の和
  s＝（a＋b）＊h／2．0；    // 正しい
```

演習 4．1

2 つの整数型の変数 a と b をそれぞれ 15 と 6 で初期化して，上記 5 つの算術演算を行った結果をそれぞれ表示するプログラムを作成せよ。

4．2　代入演算子

“x＝10；”という文が「変数 x に値 10 を代入せよ」という意味になることは既に述べた。この代入という操作も一種の演算であり，ここで“＝”は代入演算子と呼ばれる演算子のひとつである。通常の代入演算子“＝”は，以下のような形式で使用する。

変数＝変数；
または，
変数＝変数 演算子 変数 または式；
（ただし，式は，変数 演算子 変数または式）

例えば，“x＝y”は「変数 x に変数 y の値を代入せよ」を意味し、“x＝1＋2；”は「変数 x に 1＋2 の溝算結果を代入せよ」という意味である。ほかにも実例を見てみよう。

```
a＝3；              正しい
a＝1＋2；           正しい
a＝b＋（c／d）；    正しい
3＝3；              誤り
a＋b＝3             誤り
```

上記の例で誤りとなっている文は，数学ではよく使用される表現である。しかし，プログラムで“＝”が代入を表すことを考えると，「値 3 に値 3 を代入する」や「式 a＋b に値 3 を代入する」など意味をなさないことが理解できるだろう。これとは反対に，数学では絶対に使用しないがプログラム中ではよく使用される表現がある。

```
x＝ x ＋1；
```

この文は一見奇妙に見えるかもしれないが，「x に x＋1 の演算結果を代入せよ」という意味になる。結果的に，x の値は 1 だけ増えることになる。

```
x＝10；        //  変数 x に値 10 を代入
x＝ x ＋1；    //  変数 x に，変数 x の値 10 に 1 を加算した結果を代入
               //  すなわち，変数 x の値は 11 になる
```

さらに，以下のような表現も可能である。

```
a＝b＝1；
a＝b＝（c＝1）＋2；
```

代入演算子が同じ文に複数含まれる場合で，特に括弧などで順番が指定されていなければ，右側，すなわち，最後の代入から実行される。したがって，初めの文は、“a＝（b＝1）；”と同等である。すなわち，「変数 b に値 1 を代入する。その結果を変数 a に代入する」という意味になる。ここで「変数 b に値 1 を代入した結果」という表現に抵抗を感じるかもしれないが，代入も演算であることを思い出していただきたい。

Java では，代入の演算結果は代入後の変数の値となる。したがって，“b＝1”の演算結果は b の値である 1 である。2 番目の文は，「変数 c に値 1 を代入する。その結果（値 1）と 2 の和（値 3）を変数 b に代入する。さらに変数 b に代入した結果（値 3）を a に代入する」という意味に

なり，結果的に変数 a と b の値が 3 に，変数 c の値は 1 になる。

4．3　式を簡略化する代入演算子

実際のプログラムでは，ある変数の値を 1 増やすというように，変数の値を使用して演算した結果を，再度，元の変数に代入するといったことがよく起こる。このような場合に便利な，演算と代入とを合わせて表現できる代入演算子がある。このような代入演算子は，一般には以下のように表現できる。

変数　演算子＝式；

例えば，“x＝x＋1；”は，算術演算子“＋”と代入演算子“＝”を合わせた代入演算子“＋＝”によって“x＋＝1；”と書くことができる[2]。以下に例を挙げる。

[2]　Java では，＝のほかに，このような代入演算子が 11 個用意されている。＊＝　／＝　%＝　＋＝　.＝　＜＜＝　＞＞＝　＞＞＞＝　&＝　∧＝　｜＝　。

```
a＋＝b；　　⇒　a＝a＋b；
a －＝ b；　⇒　a＝a－b；
a *＝ b；　 ⇒　a＝a * b；
a /＝b；　　⇒　a＝a/b；
a *＝b－c；　⇒　a＝a*（b-c）；
```

最後の例では，a＝a*b－c；にならないことに注意すること。

演習 4．2

x 大学の学籍番号は，学部，入学年，連続番号を表す 7 桁の数字に，その 7 桁の数字から作り出す誤り検出のためのチェックデイジットを加え 8 桁の数字になっている。例えば，学籍番号 4930123 のチェックデイジットは以下のように求める。

①　4*9＋9*8＋3*7＋0*6＋1*5＋2*4＋3*3 を求める。
②　求めた値を 11 で割ったときの余りを求める。
③　さらに 10 で割ったときの余りがチェックデイジット。

式を簡略化する代入演算子を用いて，学籍番号 4930123 のチェックデイジットを求めよ。

4．4　インクリメント演算子とデクリメント演算子

道端でカウンタを使って交通量を調べたり，残りの数をチェックしながら景品を配るなど，日常生活でも 1 つずつ数えることがよく起こる。プログラム中でも，1 ずつカウントアップしたりカウントダウンする操作が必要になることがある。このための専用の演算子として，インクリメント（increment）演算子とデクリメント（decrement）演算子がある。

インクリメント演算子は，“＋＋”で変数の値を 1 だけ増加させる。

```
a＋＋；または　　＋＋a；
```

デクリメント演算子は，“－－”で変数の値を 1 だけ減少させる。

a--；　または　　--a；

もちろん，インクリメント演算子やデクリメント演算子を使用する代わりに“a＋＝1；”や“a＝a－1；”のように書くこともできる。

インクリメント演算子やデクリメント演算子を，変数の前に書いたときと後ろに書　いたときは，変数の値を 1 だけ増減させるという点では同である。しかし，式の中で使用した場合は，異なる部分も出てくるので注意を要する。変数の前に書いた場合（前置，prefix）は，変数を増加あるいは減少させて，その結果を式の中での値として使用する。変数の後ろに書いた場合は，まず変数の値を変化させる前の値を式の中で使用し，それから変数を増加あるいは減少させる。

インクリメント演算子の場合を例に，この相違を見てみる。

```
a＝5；
b＝＋＋a；
```

上記のコードでは,先に a がインクリメントされその後の値が b に代入される。すなわち，実行後は，a も b も 6 になる。

```
a＝5；
b＝a＋＋；
```

上記のコードでは，インクリメントされる前の a の値が b に代入されその後，a はインクリメントされる。すなわち，実行後は，a が 6 に，b が 5 になる。

演習 4．3
a＝3；，b＝a＋＋；，c＝＋＋a；の順番で計算した後の a，b，c の値はどうなるか考察しなさい。実際にプログラムを作成して結果を確かめよ。
a＝3；，b＝（a＋＋）；，c＝（＋＋a）；の順番ではどうなるか？

4．5　関係演算子

2 つの変数や式の値を比較して，その関係が成り立つが成り立たないかを求める演算子を関係演算子と呼ぶ。以下の 6 つがある 3。

＜　　より小さい
＞　　より大きい
＜＝　以下
＞＝　以上
＝＝　等しい
！＝　　等しくない

関係演算の結果は論理型（boolean 型）で，関係が成立すれば true，成立しなければ false になる。実際のプログラムで関係演算子を使ってみよう。

```
1:public class TestRelationOp{
2: public static void main (String argv[]){
3:   boolean isLessThan, isGreaterThan, isEqualTo, isNotEqualTo;
4:   int i = 2;
5:   int j = 3;
6:
7:   isLessThan = i < j;
8:   isGreaterThan = i > j;
9:   isEqualTo = i == j;
10:   isNotEqualTo = i != j;
11:
12:   System.out.println("i =" + i + ", j = " + j);
13:   System.out.println("i < j : " + isLessThan);
14:   System.out.println("i > j : " + isGreaterThan);
15:   System.out.println("i == j: " + isEqualTo);
16:   System.out.println("i != j: " + isNotEqualTo);
17: }
18:}
```

プログラム４．１　関係演算子（TestRelationOp.java）

プログラム 4．1 では，2 つの int 型の変数 i，j の値の関係を調べ，表示する。関係を調べた結果を代入するための boolean 型の変数を用意し（3 行目），例えば 7 行目の“isLessThan＝i＜j；”では，i が j より小さいかどうかを関係演算子＜を使用して計算し，その結果を変数 isLessThan に代入している。プログラム中でも使用されているが，“＝”は代入演算子であり，“＝＝”は関係演算子であることに十分注意を払う必要がある。実行結果を以下に示す。

```
i＝2，j＝3
i ＜ j : true
i ＞ j : false
i＝＝j : false
i ！＝j : true
```

4．6　論理演算子

論理演算子の説明に入る前に，論理演算について簡単に触れておこう。論理演算は論理型，つまり，正しい（真，true）か誤っているか（偽，false）の 2 つの値をとる。

例えば，今日は 10 月 10 日で，秋晴れの 1 日であるとしよう。

今日は晴れだ。	真（true）
今日は 8 月だ。	偽（false）
今日は 8 月で，かつ晴れだ。	偽（false）
今日は 8 月か，または晴れだ。	真（true）
今日は 8 月ではない。	真（true）

上の文章で「かつ」，「または」，「ではない」の部分がそれぞれ論理演算に相当する。論理演算にはいくつか種類がある。

（1）論理積 AND
2 つの主張が両方とも真のときに真となる演算で，図 4．1（a）の斜線部分が真である。

true AND true ⇒ true,　true AND false ⇒ false,　false AND false ⇒ false

（2）論理和 OR
2 つの主張のどちらかが真のとき真となる演算で，図 4．1（b）の斜線部分が真である。

true OR true ⇒ true,　true OR false ⇒ true,　false OR false ⇒ false

（3）排他的論理和 XOR
2 つの主張のどちらかが真で他方が偽であるとき真となる演算で，図 4．1（c）の斜線部分が真である。

true XOR true ⇒ false,　true XOR false ⇒ true，false XOR false ⇒ false

（4）否定 NOT
主張が真のとき偽，偽のときに真となる演算で，図４．１（d）の斜線部分が真である。
NOT true ⇒ false,　NOT false ⇒ true

Java では，以下の論理演算子を用意している。

演算	演算子	例
論理積	＆＆	a＆＆b
	＆	a＆b
論理和	||	a||b
	|	a | b
否定	！	！a

プログラム 4．2 に，論理演算を用いた例を示す。

```
1:public class TestLogicalOp{
2: public static void main (String argv[]){
3:   boolean a = true;
4:   boolean b = false;
5:   System.out.println("a && (!b) = "   + (a && (!b)));
6: }
7:}
```

プログラム４．２　　論理演算子（TestLogicalOp.java）

論理演算は左から右に実行される。例えば論理演算（a＜b）＆＆（b＜c）は，最初に a＜b を評価し，次に b＜c を評価する。もし，a＜b が false なら，b＜c の演算結果とは無関係に，全体の演算結果は false になることが分かる。このような場合，b＜c の評価は省略される。これを短絡的な評価と呼ぶ。論理積＆＆と||は，可能な場合は短絡的な評価を行う。これに対し

て，論理積&や論理和|は短絡的な評価は行われず，演算子の両側が常に評価される。

演習 4．4
　論理型の変数 a，b を宣言し，それぞれ true，false に初期化する。それらの論理積，　論理和を，以下のような形式で表示するプログラムを作成せよ。

　　a&&b＝（結果の出力）　　a||b＝（結果の出力）

4．7　ビット演算子とシフト演算子
　整数型について，整数を表現しているおのおののビットを直接扱うことのできるビット演算子が用意されている。以下の演算は，ビットごとの演算である。

演算	演算子[4]	例
論理積	&	a&b
論理和	｜	a｜b
排他的論理和	∧	a∧b
否定（反転）	～	～a

　また、これらに加えて，ビットパターンを左右にずらす（シフトする）シフト演算子も用意されている。

[4]　演算子&と｜は、boolean 型に対しては論理演算子として，整数型に対してはビット演算子として機能する。

演算	演算子	例
左シフト	＜＜	a＜＜b
符号付右シフト	＞＞	a＞＞b
符号なし右シフト	＞＞＞	a＞＞＞b

例えば、ビット列“0001”を左にシフトすると“0010”になる。また，“0010”を右シフトすると“0001”になる。整数の表現では，一番左側のビットは符号ビットであった。右シフトした場合に，符号ビットと同じビットが左から入る。右シフトが符号付き右シフトで，左から常に 0 がる右シフトが符号なし右シフトである。
　以下に，ビット演算とシフト演算の例を具体的に示す。

a	⇒	0x00ff	0000 0000 1111 1111
～a	⇒	0x00ff	1111 1111 0000 0000
b	⇒	0xf0f0	1111 0000 1111 0000
a&b	⇒	0x00f0	0000 0000 1111 0000
a|b	⇒	0xf0ff	1111 0000 1111 1111
a∧b	⇒	0xf00f	1111 0000 0000 1111
a＜＜1	⇒	0x01fe	0000 0001 1111 1110
a＞＞1	⇒	0x7f80	0111 1111 1000 0000
a＞＞＞1	⇒	0xff80	1111 1111 1000 0000

プログラム 4．3 に，ビット演算子を用いたプログラムの例を示す。このプログラムは，int

型の変数 a，b について，ビット演算の結果を 16 進数と 2 進数で表示する。基本的には，変数 a と b についてビット演算と表示を繰り返す。ビット論理積を計算して表示する部分（12～14 行目）を例にとると，まず，a&b の演算結果を変数 r に代入し（12 行目），その結果を表示する。その際に Integer．ToHexString()（13 行目）や Ingeter.toBinaryString()（14 行目）を使用して，r の値を 16 進数表記の文字列や 2 進数表記の文字列に変換する。

```
1:public class TestBitOp{
2: public static void main (String argv[]){
3:   int a = 0xffff0000;
4:   int b = 0xff00ff00;
5:   int r;
6:
7:   System.out.println("a                    =" +Integer.toHexString(a) +
8:                              "¥t" + Integer.toBinaryString(a));
9:   System.out.println("b                    =" +Integer.toHexString(b) +
10:                             "¥t" + Integer.toBinaryString(b));
11:
12:   r = a & b;       //ビット論理積
13:   System.out.println("a & b     =" +Integer.toHexString(r) +
14:                             "¥t" + Integer.toBinaryString(r));
15:
16: r =a | b;          //ビット論理和
17:   System.out.println("a | b     =" +Integer.toHexString(r) +
18:                             "¥t" + Integer.toBinaryString(r));
19:
20    r = a ^b;          //排他的論理和
21:   System.out.println("a ^ b     =" +Integer.toHexString(r) +
22:                             "¥t" + Integer.toBinaryString(r));
23:
24:   r = ~b;          //否定
25:   System.out.println(" ~b       =" +Integer.toHexString(r) +
26:                             "¥t" + Integer.toBinaryString(r));
27:  }
28:}
```

プログラム４．３　ビット演算子（TestBitOp.java）

4．8　混合演算とキャスト

2 つの変数の間で演算をするときに，それぞれの変数のデータ型が異なっていたらどうなるのであろう。例えば，short 型で表現できる値は int 型でも表現できるが，逆に int 型で表現できる値のすべてを short 型で表現できるわけではない。単純に言えば，データ長が長いほど値の守備範囲が広い。このように，データ型の間にはランクがある。

byte＜short＜int＜long＜float＜double

型の変換は，自動的に行われる場合と，明示的に指定して行う場合がある。算術演算でデー

タ型が混在する場合がよくある。このような演算を，ここでは混合演算と呼ぶ。混合演算では，データ型を変換してランクを合わせる操作が行われる。特にランクが上のデータ型への変換を格上げと呼ぶ。byte 型から long 型までの格上げでは精度に変化はないが，long 型から浮動小数点型への格上げでは精度が悪くなることがあるので，場合によっては注意が必要である。

混合演算では，以下のように自動的に型の変換が行われる。

●演算子の片側が double 型なら他方を double 型に変換する
●そうでなければ，演算子の片側が float 型なら他方を float 型に変換する
●そうでなければ，演算子の片側が long 型なら他方を long 型に変換する
●そうでなければ，演算子の両側を int 型に変換する

したがって，byte 型の変数 b，int 型の変数 i，j，float 型の変数 f，double 型の変数 d とすると，それぞれの演算結果は次のようなデータ型になる。

d＋f　＝＞　double 型
i＋j　　⇒　int 型
i＋b　　＝＞　int 型
b＋b　⇒　　int 型

変数の型を一時的に変更したいことが起こる場合がある。Java ではキャストにより強制的に型変換を行うことができる。キャストの形式は以下の通りである。

（型）式

例えば，short 型の変数 s を例にとると以下のようになる。

（int）s　　int 型へのキャスト
（float）s　　float 型へのキャスト
（double）s　　double 型へのキャスト

プログラム 4．4 では，double 型の変数 a をいったん int 型にキャストして，再度 double 型の変数 b に代入している。このような演算の結果，変数 b の値はどうなるであろうか。（int）a が実行されると，この式の値は整数の 3 になる。しかし，b への代入ではデータ型は左辺に合わせられ（格上げされ），b には double 型の 3．0 が代入される。

```
1:public class TestCast{
2: public static void main (String argv[]){
3:   double a, b;
4:   a = 3.14159265;
5:   b = (int)a;
6:
7:   System.out.println("a = " + a);
8:   System.out.println("b = " + b);
9: }
10:}
```

プログラム４．４　キャスト（TestCast.java）