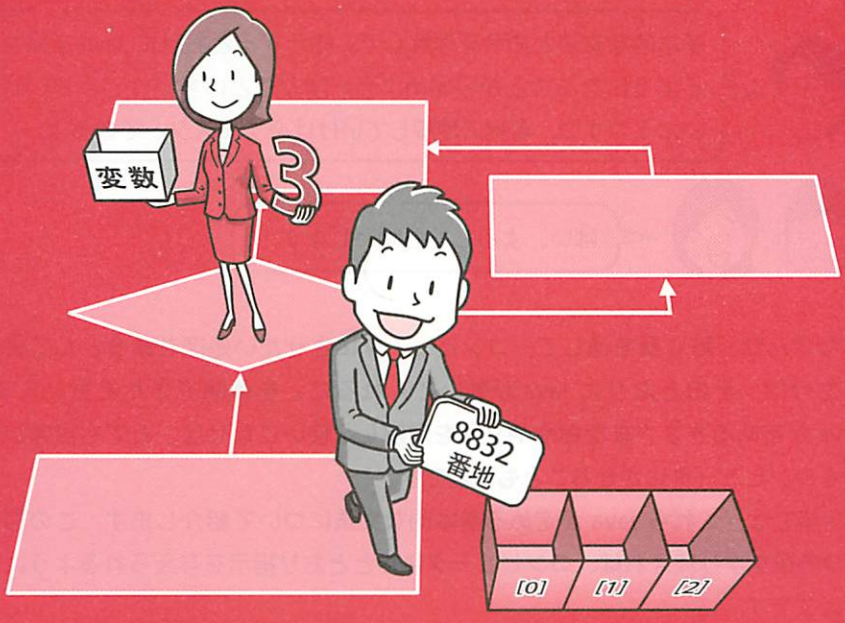
第 I 部

ようこそ Java の世界へ



Java プログラミングことはじめ

まだ 1 時間も経っていないのに、Java のプログラムを作って動かせちゃったね♪

そうだね。あ、菅原さん。Java にはスゴい命令とか便利な機能がいっぱいあるんですよね？！　RPG のアイデアがいっぱいわいてきちゃって、ボク、一日も早く Java をマスターしたいんです！

まぁあせらずに。せっかくだから、ちゃんと腰を据えて学ぼうじゃないか。しっかり基礎を固めれば、立派な RPG が作れるようになるよ。

まずは何から学んだらいいのかしら。

基本的な文法と命令を理解して、指示するとおりに Java を操れるようになることから始めよう。学ぶことは多いけど、ていねいに 1 つずつ、着実に理解していけば必ずマスターできるよ。

はい、よろしくお願いします！

みなさんは、第 0 章を通して、コンピュータにさせたい処理を命令として書いておけば、そのとおりに Java が動いてくれることを体験できたと思います。Java が定めるさまざまな命令や文法を理解し、使いこなせば、とても複雑な処理をコンピュータにさせることも可能です。

第 I 部では、これら Java が定める基本的な文法について紹介します。この部の 6 つの章を学び終えれば、コンピュータにひととおり指示を与えられるようになるはずです。

第1章

プログラムの書き方

Javaでは、プログラムの書き方に関するさまざまなルールが
定められています。
なかでも、作るプログラムの内容や規模に関係なく、
必ず使うことになる基本的なルールはとても重要です。
まずはそれら基本的な文法をしっかりおさえることから学習を始めましょう。

CONTENTS

1.1 Java開発の基礎知識

1.1.1 開発の流れ

では、まず Java 開発の流れを再確認していこう。

はい。3 ステップでしたよね。

プログラミング体験で繰り返し行ったように、Java の開発は、①ソースコードの作成、②コンパイル、③実行、この 3 ステップで行います（図 1-1）。では、それぞれのステップを詳しく解説していきましょう。

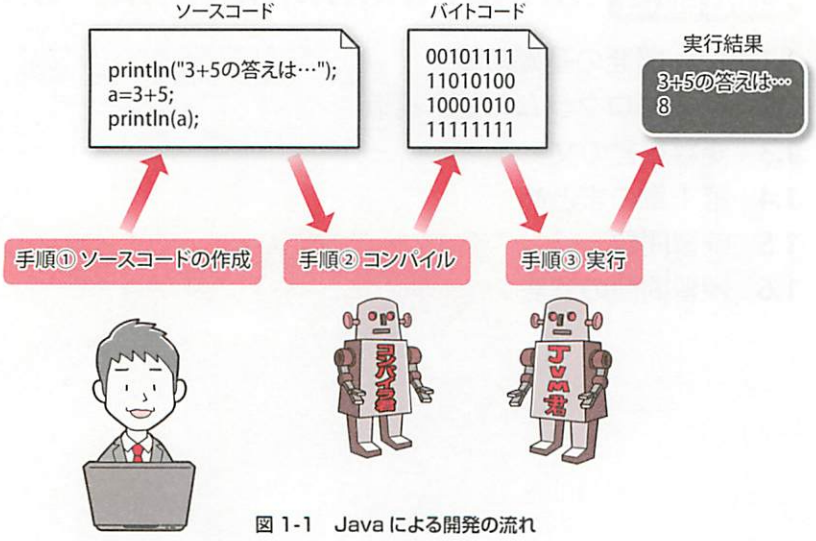


図 1-1　Java による開発の流れ

手順① ソースコードの作成

最初にJavaが定める文法に従ってコンピュータへの命令を記述していきます。「public class ～」のような記述で、人が読める状態のプログラムを**ソースコード**(source code)、または単にソースと言います。

dokojavaの画面で入力したものがソースコードなのね。

手順② コンパイル

コンピュータは、その心臓部であるCPUがプログラムを解釈しながら指示どおりに動きます。しかし、CPUは私たちが書いたソースコードを直接、読んで動くことはできません。CPUは**マシン語**(machine code)と呼ばれるCPUが理解できる言葉で書かれたプログラムしか実行できないのです。

そこで私たちは、まずソースコードに対して**コンパイル**という処理を行って**バイトコード**(byte code)という状態に変換します。この際、ソースコードの文法チェックも行われ、もし誤りがあればコンパイルは失敗し、誤りの箇所が表示されます。この一連の処理は**コンパイラ**(compiler)と呼ばれる変換ソフトウェアが担当します。

バイトコードやマシン語は1と0が複雑に並んでいるもので、人間には、とても読めない言葉なんだ。

手順③ 実行

コンパイルが無事完了したら、**インタプリタ**(interpreter)と呼ばれるソフトウェアに対してバイトコードの実行を指示します。インタプリタは**JVM**(Java Virtual Machine)というしくみを内部に持っており、バイトコードを少しずつ読み込みながら、それをマシン語に変換してコンピュータのCPUに送ります。こうしてコンピュータはソースコードで指示したとおりに動作するのです。

ボクたちが書いたソースコードは途中で2回も変換されてから実行されるんですね。

1.1.2 開発環境の整備

Java プログラミングを行うために、私たち開発者はコンパイラとインタプリタを準備する必要があります。その具体的な方法は次の 2 つです。

①自分の PC にコンパイラとインタプリタをインストールする

インターネットから Java のコンパイラやインタプリタをダウンロードして、それを自分の PC にインストール（導入）します。Java 開発者の多くは、この方法を用いますが、初心者には少し難しい手順が含まれています。

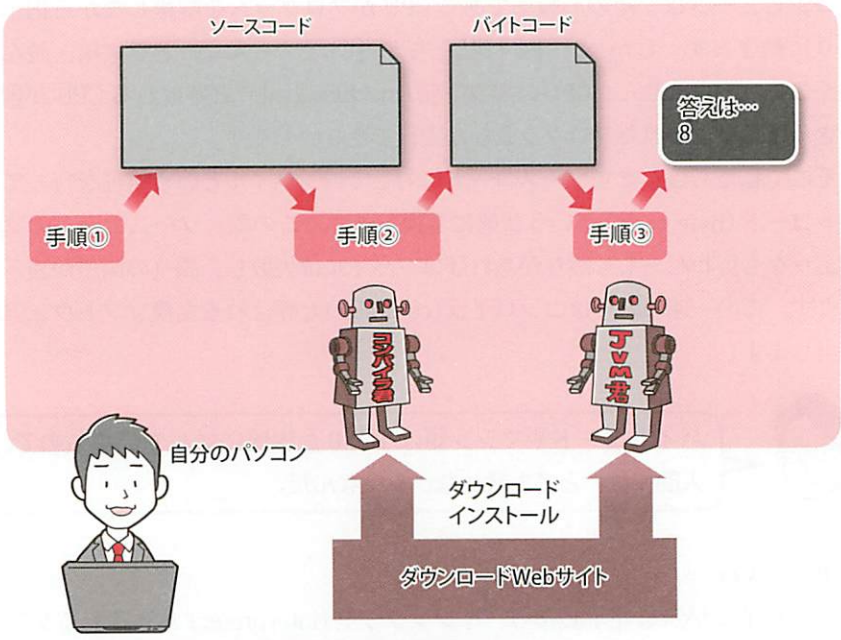


図 1-2　自分の PC に構築した開発環境による開発

② dokojava サーバにコンパイルと実行をさせる

インターネット上に準備されている dokojava サーバには、コンパイラとインタプリタがすでに導入されています。開発者がブラウザ画面からソースコードを送れば、サーバ上でコンパイルと実行が行われ、その結果が Web ブラウザの画面に戻されます。

難しいセットアップなしでJavaのプログラミングが可能なメリットがありますが、さまざまな理由からJavaの機能を一部、利用できない制約があります。

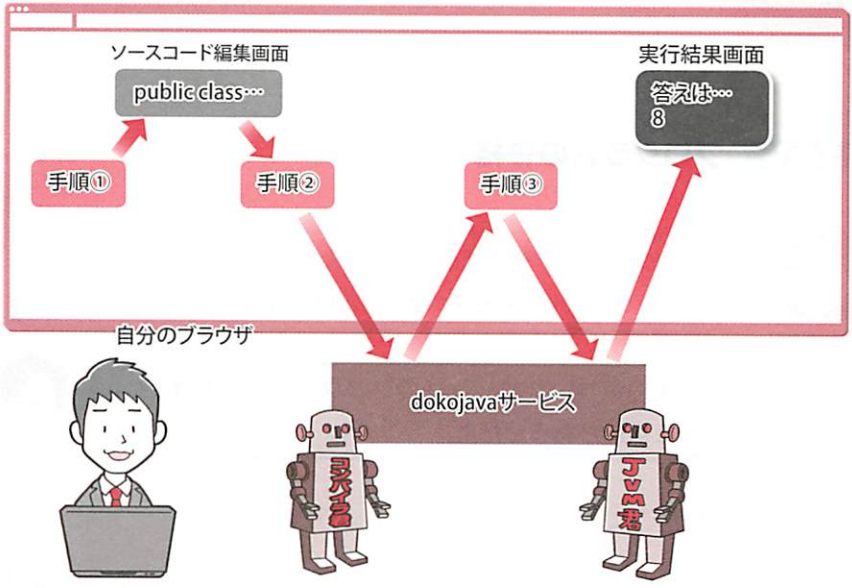


図 1-3　dokojavaを用いた開発

この本で取り扱う範囲のプログラムを学習用に作成して動かす目的であれば、どちらの方法でも構いません。dokojavaを利用して学習を進める方は、このまま次節以降に進んでください。

自分のPCにコンパイラとインタプリタを導入して学習を進めたい方は、先に付録A「JDKを用いた開発」を参考にインストールを行ってください。

開発の流れがわかり、開発環境も揃いました。
あとはソースコードを書くだけですね！

次節からはソースコードの記述方法を解説していくよ。

1.2 Javaプログラムの基本構造

1.2.1 プログラムの骨格

プログラミングを体験して感じましたけど、いろいろ命令を書いていくのは「真ん中の部分」だけなんですね。

よく気づいたね。Java のプログラムには基本的な構造があるんだよ。

まず Java プログラムの全体像を見てみましょう。

リスト 1-1

Main.java

```
1  public class Main {
2    public static void main(String[] args) {
3      System.out.println("RPG:スッキリ魔王征伐");
4      System.out.println("Ver.0.1 by 湊");
5      System.out.println("＜ただいま鋭意学習・制作中＞");
6      System.out.println("プログラムを終了します");
7    }
8  }
```

処理・命令の部分

Java のソースコードには、波カッコ {…} で囲まれた部分が多く登場します。この波カッコで囲まれた部分を**ブロック**（block）と呼びます。外側のブロックはクラスブロック（class block）、内側のブロックはメソッドブロック（method block）と呼ばれ、Java のソースコードは必ずこれらのブロックによる二重構造を持っています。

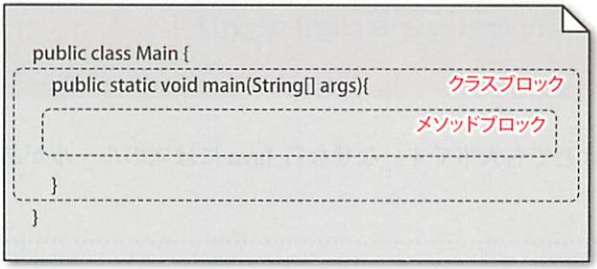


図 1-4　Java のソースコードのブロック構造

私たちが「コンピュータに対する指示・命令」を記述していくのはメソッドブロックの中です。それより外側（最初の２行と最後の２行）は、どのようなプログラムを開発する場合も、あまり変わらない「お決まりのパターン」です。

図 1-5　外側の部分はお決まりのパターン

お手紙を書くときに、本文の前後に「時候の挨拶」と「結語」を書くのと似ていますね。

外側部分は毎回ほとんど同じだから、何度も書いて覚えてしまおう。

ソースコードの外側部分はお決まりパターンとはいえ、どんなプログラムを開発するときも「毎回まったく同じ記述でいい」わけではありません。１行目にある「public class」の直後に書かれる単語は、このプログラムの名前を指定するものです。正式には**クラス名**（class name）といい、大文字のアルファベットで始まる名前を付けることが一般的です。

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

クラス名の指定

クラス名はとても大事です。なぜなら Java には次のルールがあるからです。

Javaソースファイルの名前

Java のソースコードを記述したファイル（ソースファイル）を保存するときには、ファイルの名前は「クラス名 .java」にしなければならない。

Java プログラムのクラス名に対するルールを次にまとめました。

MyDiary.java

```
public class MyDiary {
  public static void main(String[] args) {
    ⋮
  }
}
```

クラス名の指定

※ソースファイル名は「クラス名 .java」にする。
※クラス名はアルファベット大文字で始める。

1.2.2 プログラムの書き始め方

1.2.1 項で学んだことをまとめると、私たちは次の流れでソースコードを作っていくことになります（図 1-6）。

Java プログラムの書き始め方

①どのようなプログラムを作りたいかを考えます。
②プログラムの名前を決めます（クラス名が決まります）。

③「クラス名 .java」という名前でファイルを作ります。
④ソースコードの外側部分を記述します。
⑤ソースコードの内側部分に命令を書いていきます。

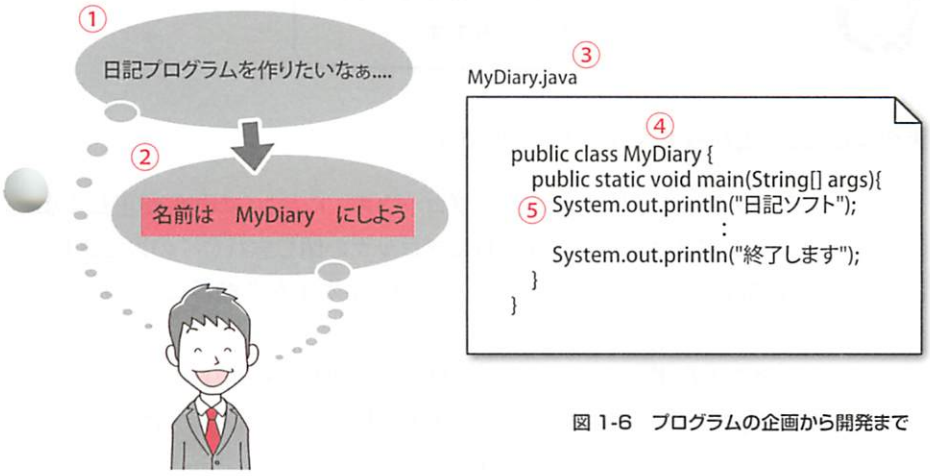


図 1-6　プログラムの企画から開発まで

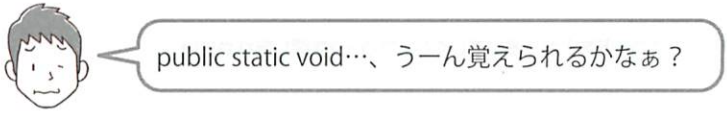
ソースコードの記述に際しては、意識すべき大切なことが３つあります。

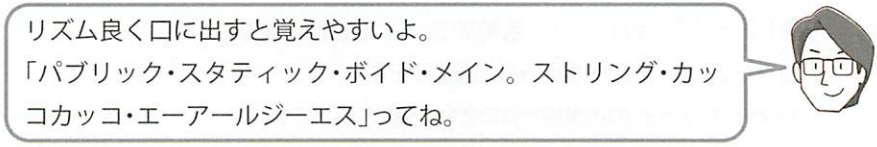
大切な意識 1：正確に記述する

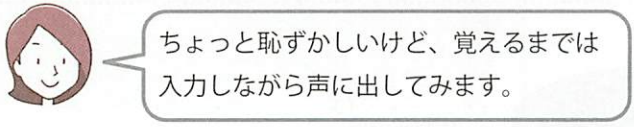
ソースコードには、さまざまな文字・数字・記号が登場します。見た目が似ていても、間違った文字を入力するとプログラムは正常に動きません。特に次の点に気をつけましょう。

- 英数字は基本的に半角で入力し、大文字小文字の違いも意識する。
- オー (o・O) とゼロ (0)、エル (l) と数字のイチ (1)、セミコロン (;) とコロン (:)、ピリオド (.) とカンマ (,) を間違えない。
- カッコ ((), {}, []) や引用符 ('、") の種類を間違えない。

特に正確な記述を求められるのはプログラムの２行目です。「public static void main (String[] args)」を一字一句間違えずにスラスラ書けるようになりましょう。







大切な意識 2:「上から下へ」ではなく「外から内へ」

プログラミングを始めてしばらくは、「ブロックの“{”と“}”の対応が正しくない」というエラーに悩まされがちです。特にブロックの閉じ忘れ(“}”の書き忘れ)に悩まされる初心者に共通するのは、「ソースコードを上から順番に記述している」ことです。

ソースを上から 1 行ずつ記述すると「①ブロックを開く→②中身を書く→③ブロックを閉じる」という手順になるので、中身を書いている間に「ブロックを閉じる必要があること」を忘れてしまいます。

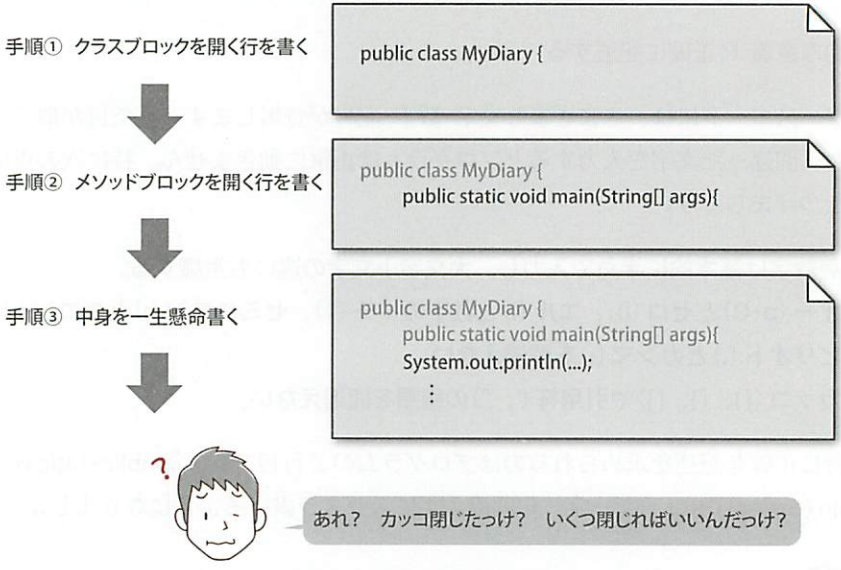


図 1-7　ソースコードを上から書いていくと、ブロックを閉じ忘れやすい

このエラーを防ぐため、次の図のように「ブロックを開いてすぐに閉じる」「中身を一生懸命作る」という手順で書くようにするとよいでしょう（メソッドについては 1.2.5 項で解説します）。

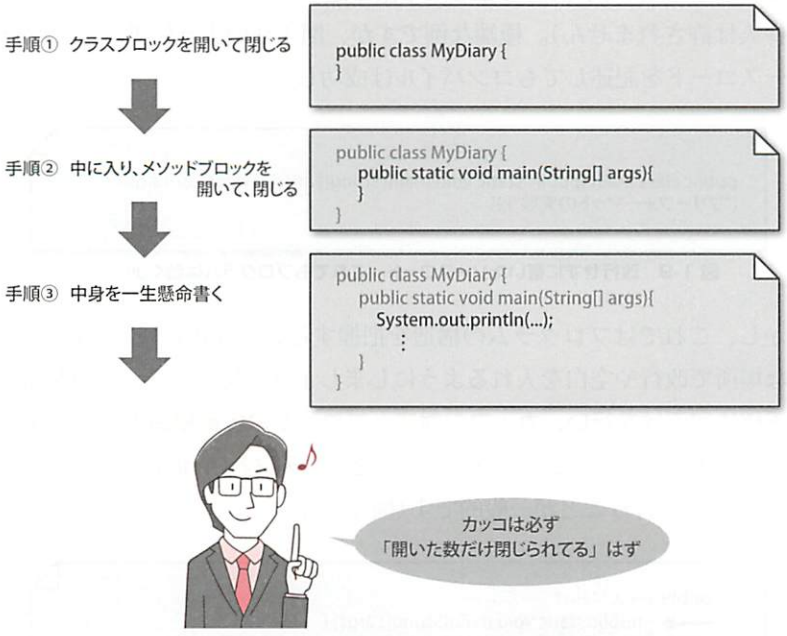


図 1-8　ソースコードを外から内へ書いていくと、閉じ忘れの失敗が減る

大切な意識 3：読みやすいコードを記述する

文法的には誤りがなくても、「人間が読みにくい煩雑なコード」や「複雑すぎて内容の理解に時間がかかるコード」では修正や改良が難しくなります。特に仕事で Java プログラムを作る人は、同僚や取引先にソースコードを見てもらうことがあるため、誰が見てもわかりやすい記述をするようにしましょう。

先輩。具体的にどういう工夫をしたら「読みやすいコード」が書けるようになりますか？

「インデント」と「コメント」を上手に活用するといいよ。

1.2.3 インデント

Java では「ソースコード中、どこに改行や空白を入れるかは基本的に自由」とされています（ただし、public などの単語が途中で切れてしまうような改行や空白の挿入は許されません）。極端な例ですが、図 1-9 のようにまったく改行せずにソースコードを記述してもコンパイルは成功します。

```
public class Main {public static void main(String[] args) {System.out.println
("フリーフォーマットの実験");}}
```

図 1-9　改行せずに書いたソースコード。これでもプログラムは動くが…

しかし、これではプログラムの構造を把握することは難しいですね。そこで、適切な場所で改行や空白を入れるようにしましょう。特に**ブロックの開始と終了では正確に字下げを行い、カッコの対応とブロックの多重構造の見通しを良くする**ことがポイントです。この字下げのことを**インデント**（indent）と呼び、キーボードの TAB キーで行うことが一般的です（図 1-10）。

```
public class Main {
    public static void main(String[] args) {
        System.out.println("フリーフォーマットの実験");
    }
}
```

図 1-10　インデントを入れたソースコード。ブロックの構造がわかりやすくなる

インデントは、でたらめに入れてはいけません。誤ったインデントはプログラム構造の読み間違い、そして致命的なエラーの原因になることもあるからです。

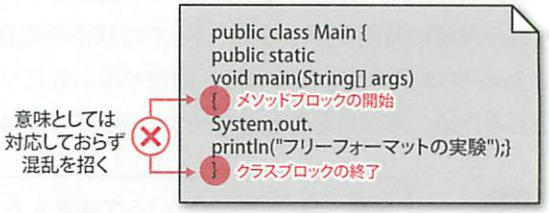


図 1-11　混乱を招くインデント

とはいっても、ちょっとぐらいインデントが変でも、まぁ動くし…いいじゃないですか？

ダメだよ。以降の章では、ブロックが4重や5重の入れ子構造になる書き方も登場するし、良くない書き方を最初に身に付けてしまうと、その後の学習効率にも大きく影響を与えてしまう。今のうちに「正確なインデント」を入れる習慣を身に付けてほしい。

1.2.4 コメント

今は10行程度だけど、大きいコードになると「何行目で何をやっているか」わからなくなってしまいそうですね。

大丈夫。ソースコードの中には解説文を書き込めるんだ。もちろん日本語もOKだよ。

よりプログラムを読みやすくするため、ソースコード中に解説文を書き込むこともできます（次ページ図1-12）。この解説文を**コメント**（comment）といい、プログラムのコンパイル時と実行時には無視されます。人が読むためだけに書かれるものですから日本語で書いて構いません。

コメント文①　複数行コメント

```
/* コメント本文（複数行でも可）*/
```

コメント文②　単一行コメント

```
// コメント本文（行末まで）
```

/* から */まではコメント

```
/* サンプルプログラム Main
   開発者: 菅原　　作成日: 2011年4月
*/
public class Main {
    // ここからmainメソッド
    public static void main(String[] args) {
        int age; // 年齢を入れる箱
        age = 20;
        System.out.println("私は" + age + "歳");
    }
}
```

// から行末まではコメント

図 1-12　コメントはソースコードのあらゆるところに記述できる

1.2.5 main メソッドの中身

残るは「ソースコードの内側部分」の記述方法だね。

どんなルールに従ってプログラムを書いていけばいいのかな？

1.2.1 項で学んだように、計算や表示など Java の命令を書いていく場所はメソッドブロックの内部です。ソースコードの 2 行目に「main」の表記が出てくることから、この部分は main メソッドとも呼ばれます（図 1-13）。

main メソッドの中には、文（statement）を順番に書いていきます。プログラムの実行時には、文は上から順に 1 行ずつ処理されていきます。

文の末尾には必ずセミコロン（;）を付けるのが Java のルールだ。忘れやすいから注意してほしい。

図 1-13 の MyDiary のソースコードの例にあるように、main メソッドの中にはさまざまな文を書くことができます。Java には、とても多くの種類の文が存在しますが、図 1-14 に示した 3 種類に分類して考えると理解しやすいでしょう。

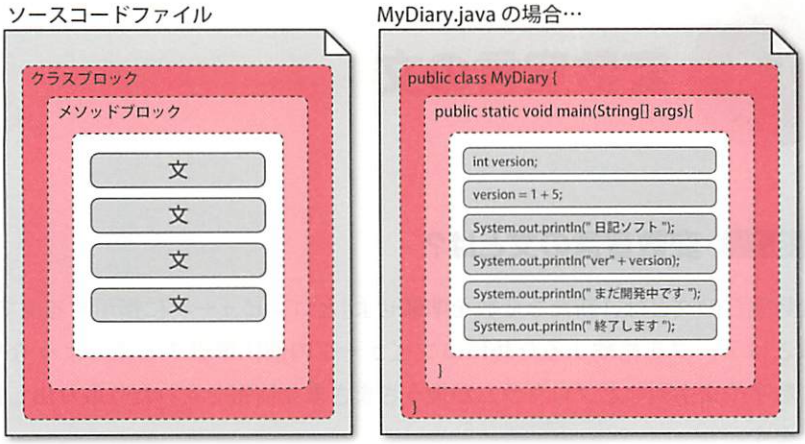


図 1-13　main メソッドのブロック内に「文」を書いた例

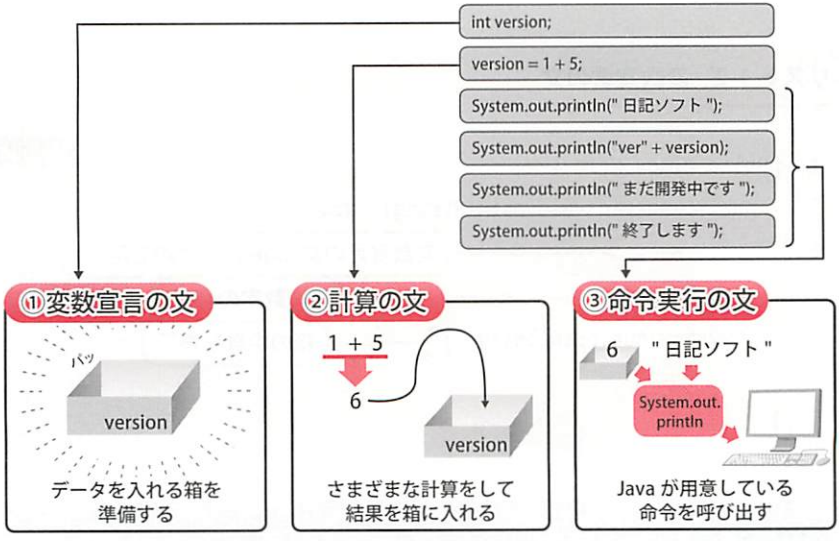


図 1-14　文は 3 種類に分けることができる

次節では、まず「①変数宣言の文」について学びましょう。

1.3 変数宣言の文

1.3.1 変数宣言の文とは?

変数宣言の文とは、「新たな変数を準備せよ」とコンピュータに指示する文です。変数とはデータを格納するためにコンピュータ内部に準備する箱のようなもので、数字や文字列などプログラムが扱うさまざまな情報を入れたり取り出したりできます。

変数の実体はコンピュータのメモリにある区画です。変数に値を入れると実際にはメモリに値が書き込まれます。では、実際に変数を利用している例を見てみましょう。

リスト 1-2　変数宣言の文

Main.java

```
1  public class Main {
2    public static void main(String[] args) {
3      int age;
4      age = 30;
5      System.out.println(age);
6    }
7  }
```

変数宣言の文（age という箱を用意）

箱に数字の「30」を入れる

箱の中身を表示

実行結果

```
30
```

このプログラムでは変数 age に値「30」を入れ、それを取り出して画面に表示しています。このように変数に値を入れることを「代入」、取り出すことを「取得」と呼びますが、どちらも変数を宣言した後にしかできません。

変数を使いたかったら、まずは「変数宣言の文」を使って宣言する必要があるんだね。

変数を宣言するときには、変数名（データを入れる箱の名前）と型（データを入れる箱の種類や大きさ）の２つを必ず指定する決まりになっています。

変数宣言の文

```
型　変数名;
```

型とは「変数に入れることができるデータの種類」のことです。たとえばリスト 1-2 で使われている int は「整数」を表す型です。この型で宣言された変数 age には整数しか入れることはできず、小数や文字列を入れることはできません。

なるほど！ ちなみに int のほかに、どんな型があるんですか？ 変数に付ける名前に決まりはあるんですか？　それと…。

まぁ落ち着いて。まずは「変数に付ける名前」について次項で説明しよう。

1.3.2 変数の名前

変数を宣言する際、私たちは変数に名前を付ける必要があります。Java プログラミングでは変数以外にも名前を付けることがありますが、それらの名前として使える文字や数字の並びのことを**識別子**（identifier）と呼びます。

名前を何にするかは基本的にプログラマの自由ですが、通常はアルファベット、数字、アンダースコア「_」、ドル記号「$」などを組み合わせて作ります。ひらがなや漢字を含めることも可能ですが、推奨されません。そのほかにも次ページのルールや慣習があるので注意してください。

■禁止されている単語を使ってはならない

Java では、「そもそも名前として使ってはいけない単語」とされている**予約語**（keyword）が約 50 個あります。これまでにも登場した int や void、public、static などは予約語ですので、これらを変数名として利用することはできません。

■すでに利用している変数名を再度使ってはならない

すでに変数 name を宣言しているのに、再び変数 name を宣言してはいけません。2 つの変数が区別できなくなってしまうからです。

■大文字・小文字・全角・半角の違いは区別される

大文字／小文字、全角／半角の違いは人間にはささいなものですが、Java では完全に区別されます。たとえば変数 name と変数 Name は別のものとして扱われますので注意してください。

■小文字で始まるわかりやすい名前を付けることが望ましい

慣習的に、変数には小文字で始まる名詞形の名前を付けます。また、複数の単語をつなげて変数名にする場合、2 つ目以降の単語の先頭は大文字にします。たとえば「myAge」などです。

また、格納される情報の内容を想像しやすく、わかりやすい変数名が望ましいでしょう。第 3 章で登場するループ変数など一部例外はありますが、a や s のような 1 文字の変数名は避けるようにしましょう。なお、本書では紙面スペースの都合により短い変数名を用いる場合があります。

基本的に自由でも、好き放題に名前を付けていいわけではないんですね。

Java の予約語一覧

以下は Java の予約語ですので、変数名には使えません。以降、本書のリストでは、予約語を「long」のように色付きで表します。

abstract、assert、boolean、break、byte、case、catch、char、class、const、continue、default、do、double、else、enum、extends、final、finally、float、for 、if、goto、implements、import、instanceof、int、interface、long、native、new、package、private、protected、public、return、short、static、strictfp、super、switch、synchronized、this、throw、throws、transient、try、void、volatile、while

1.3.3 データ型

変数宣言に使う「型」については、どんな種類があるんですか？
いっぱいあったら覚えるのが大変だなぁ…。

まずは次の９つを覚えておけば大丈夫だよ。

プログラムで扱うことができるデータの種類のことを、**データ型**(data type)または単に**型**と言います。Java には多くの型が準備されていますが、今は次の９つだけを覚えておけばよいでしょう。

分類	型名	格納するデータ	変数宣言の例		利用頻度
整数	long	大きな整数	long worldPeople;	//世界の人口	△
	int	普通の整数	int salary;	//給与金額	◎
	short	小さな整数	short age;	//年齢	△
	byte	さらに小さな整数	byte glasses;	//所持する眼鏡の数	△
小数	double	普通の小数	double pi;	//円周率	○
	float	少しあいまいでもよい小数	float weight;	//体重	△
真偽値	boolean	trueかfalse	boolean isMarried;	//既婚か否か	○
文字	char	1つの文字	char initial;	//イニシャル1文字	△
文字列	String	文字の並び	String name;	//自分の名前	◎

図 1-15　代表的な９種類のデータ型

それでは、９つの型を４つのグループに分けて１つずつ紹介しましょう。

■整数を格納できる 4 つの型（long、int、short、byte）

long 型、int 型、short 型、byte 型の変数には整数を代入できます。

```
long worldPeople; worldPeople = 6900000000L;
int salary; salary = 152000;
short age; age = 18;
byte glasses; glasses = 2;
```

末尾の L は第 2 章で解説します

これら 4 つの型は箱の大きさ（コンピュータ内部で準備されるメモリの量）に違いがあります。そのため、代入できる値の範囲に図 1-16 の制限があります。

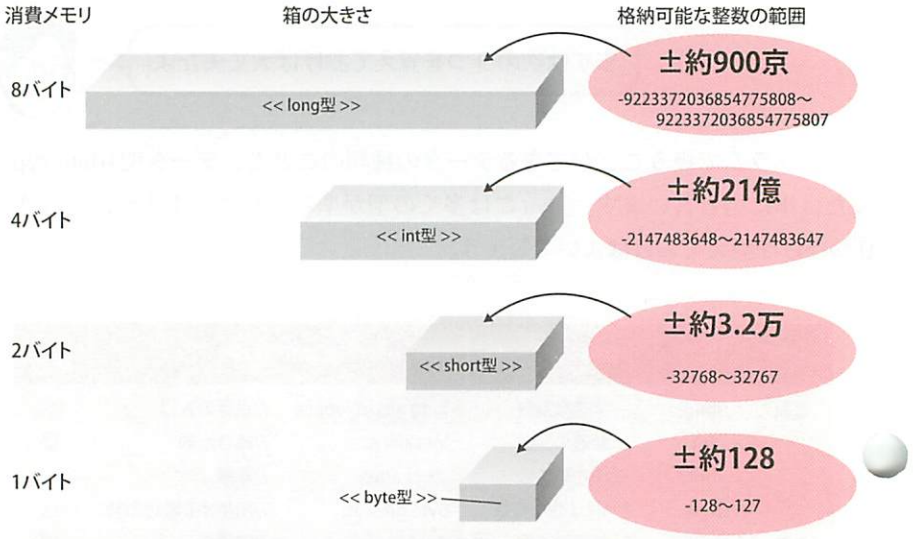


図 1-16　long 型、int 型、short 型、byte 型に代入できる整数の範囲

たとえば byte 型の変数を宣言した場合、消費するメモリは 1 バイトだけなので、これには－128 ～ 127 までの数字しか代入できません。

一方、long 型の変数を宣言した場合、8 バイトのメモリを消費しますが、－9223372036854775808 ～ 9223372036854775807 という、とても大きな整数を代入できます。

どれを使うのがベストか迷っちゃいそう。年齢なら byte 型で十分だし…いや、一応 short にしておいたほうが安全かしら…。

特殊な場合を除けば、常に int を使っておけば大丈夫だよ。

最近のコンピュータは多くのメモリを搭載しているため、これら 4 つの型を厳密に使い分ける必要があるケースはまれです。また、short や byte より int のほうが高速に処理できるコンピュータも多いので、**整数を代入したい場合、通常は int 型**を使えば問題ありません。

■小数を格納できる 2 つの型 (double, float)

double と float は「3.14」や「−15.2」といった小数を含む数値を代入するための型です。コンピュータの内部では小数を「浮動小数点」という形式で管理していることから**浮動小数点型**(floating point type) と総称されることもあります。具体的には以下のように用います。

```
double height; height = 171.2;
float weight; weight = 67.5F;
```

末尾の F は第 2 章で解説します

double のほうが float より多くのメモリを消費しますが、より厳密な計算を行うことができます。そのため、**特別な事情がない限り double 型を使用します。**

ここで 2 人に覚えておいてほしいことがあるんだ。忘れると大事故につながる大切なことだよ。

実は、浮動小数点方式には「**真に厳密な計算ができない**」という弱点があります。つまり、計算を行った際にわずかな誤差が発生することがあるのです。通常は無視できるほど小さな誤差ですが、それが積み重なると大きな問題になることもあります。そのため、誤差が許されない計算、特に**お金の計算に double や float を使ってはいけません。**

■ YES か NO かを格納できる boolean 型

boolean 型は、「YES か NO か」「本当か嘘か」「裏か表か」「成功か失敗か」といった二者択一の情報を代入するための型です。肯定的情報を意味する true、否定的情報を意味する false のどちらかの値のみを代入することができます。

```
boolean isMarried; isMarried = true;
boolean result; result = false;
```

意味：結婚している

意味：結果は失敗

なお、true は真、false は偽という意味を持つので、boolean 型のことを真偽値型と言う場合もあります。

■ 1 文字だけを格納できる char 型、文字列を格納できる String 型

char 型は全角・半角を問わず「1 文字」を代入できる型です。一方、String 型は文字列（0 文字以上の文字の集まり）を代入できる型です。具体的には次のように使います。

```
char gender; gender = '男';
String name; name = "すがわら";
```

この例にあるように、ソースコードに「文字」データを記述する場合は引用符（'）で囲みます。そして「文字列」データを記述する場合は二重引用符（"）を使います。このため「char gender; gender = " 男 ";」とするとコンパイルエラーになります。

また、全角の引用符を使わないように気をつけよう。日本語の入力をした後で、ついつい後ろの「'」や「"」も全角にしてしまうミスがよくあるよ。

1.3.4 変数の初期化

何のために変数宣言をするかといえば、「変数にデータを入れたいから」です。

```
int age;
age = 22;
```

変数宣言の文

age に「22」を代入

この文は次のように 1 行にまとめて書くこともできます。

```
int age = 22;
```

変数宣言と代入を 1 行で行う

このように「変数を宣言すると同時に値を代入すること」を**変数の初期化**と呼びます。

変数の初期化

型 変数名＝代入するデータ;

1.3.5 定数の利用

変数には異なる値を何度でも代入できます。変数に値を入れた後、別の値を代入するとどうなるかをリスト 1-3 で見てみましょう。

リスト 1-3　変数の再代入

Main.java

```
1  public class Main {
2    public static void main(String[] args) {
3      int age = 20;
4      System.out.println("私の年齢は" + age);
5      age = 31;
6      System.out.println("…いや、本当の年齢は" + age);
7    }
8  }
```

変数 age を 20 で初期化

変数 age に再度代入している

実行結果

私の年齢は20
…いや、本当の年齢は31

この実行結果からわかるように、変数の内容は新たな値 31 で上書きされ、古い値 20 は消滅します。

変数の上書き

すでに値が入っている変数に代入をすると、古い値は消滅し、新しい値に内容が書き換わる。

しかし、プログラムを開発していると、「絶対に上書きされたくない」「内容が書き換えられたら困る」場合もあります。次のコードを見てください。

リスト 1-4　書き換えてはいけない変数の値を上書きしてしまった例

Main.java

```
1  public class Main {
2    public static void main(String[] args) {
3      double tax = 1.08;
4      int fax = 5;
5      System.out.println("5万円から4万円に値下げします");
6      tax = 4;
7      System.out.println("ＦＡＸの新価格（税込み）");
8      System.out.println(fax * tax + "万円");
9    }
10 }
```

消費税を入れた変数

fax は 5 万円

誤り！代入すべきは fax 変数

実行結果

```
5万円から4万円に値下げします
ＦＡＸの新価格（税込み）
20.0万円
```

「値下げします」って表示しながら、大幅値上げしちゃってますね。

でも、これが本物のネットショップのプログラムだったら、笑いごとじゃ済まないわよ。私もミスをしちゃいそう…。

6行目で、fax変数に代入すべきところをtax変数に代入して上書きしてしまったため、計算結果がおかしくなってしまいました。そもそも税率であるtaxは、**その内容が動作中に書き換わる必要のない変数**です。このような場合、変数taxの宣言の前に**final**という記述を加えることで書き換えを防止できます。

final付きで宣言された変数は**定数**（constant variable）と呼ばれ、宣言と同時に初期値が代入された後は、値を書き換えることはできません。

定数の宣言方法

```
final 型 定数名 = 初期値 ;
```

※定数名にはすべて大文字（「TAX」など）を用いることが一般的

先ほどのリスト1-4を修正すると次のようになります。

リスト1-5　定数の例

Main.java

```
1  public class Main {
2    public static void main(String[] args) {
```

```
3       final double TAX = 1.08;
4       int fax = 5;
5       System.out.println("5万円から4万円に値下げします");
6       TAX = 4;
7       System.out.println("ＦＡＸの新価格（税込み）");
8       System.out.println(fax * TAX + "万円");
9     }
10  }
```

定数として税率を設定

コンパイルエラーとなり誤りに気づく

これで変数宣言の文はレッスン終了だよ。

1.4 第 1 章のまとめ

この章では、次のようなことを学びました。

開発と実行の流れ

- Java の文法に従いソースコードを作成する。
- ソースコードをコンパイラでコンパイルして、バイトコードに変換する。
- インタプリタはバイトコードをマシン語に変換しながら CPU を動かす。

開発の流れと基本構造

- ソースコードはブロックによる二重構造を持っている。
- 外側部分は形式的記述であり、内側に文を並べる。
- 読みやすいソースにするためコメントとインデントを活用する。

変数宣言の文

- 変数は「型 変数名 ;」で宣言して利用する。
- 変数名は基本的に自由だが、一定の制約がある。
- 変数には代表的な 9 つの型があり、用途に合わせて使い分ける。
- final を付けて宣言された定数の値は書き換えられない。

1.5 練習問題

練習 1-1

次の文章の［　　］に入る言葉を答えてください。

Java でプログラムを開発するためには、［　ア　］と［　イ　］というソフトウェアが必要です。［　ア　］は、私たちが Java の文法に沿って記述した［　ウ　］を［　エ　］に変換してくれます。［　イ　］は内部に持っている［　オ　］のしくみを使ってこれを解釈し、マシン語に変換して CPU が実行します。

練習 1-2

画面に次のような結果を表示するソースコードを作成してください。このとき、ソースコード内で 3 を変数 a に、5 を変数 b に入れ、その掛け算の結果を変数 c に入れて、最後に変数 c を表示してください。

縦幅 3 横幅 5 の長方形の面積は、15

練習 1-3

以下に示した 5 つの値を格納するために適した型を考え、「初期化」による宣言を行うソースコードを作成してください（画面に表示する必要はありません）。なお、変数名は自由に考えて構いませんが、Java のルールに従ったものにしてください。2 つ以上の型が考えられる場合は、そのどちらでも構いません。

① true　② '駆'　③ 3.14　④ 3141592658539791
⑤ " ミナトの攻撃！ 敵に 15 ポイントのダメージを与えた。"

1.6 練習問題の解答

練習 1-1 の解答

(ア)コンパイラ　(イ)インタプリタ　(ウ)ソースコード
(エ)バイトコード　(オ)JVM

練習 1-2 の解答

以下は解答例です(おおむね合っていれば正解で構いません)。

Main.java

```
1  public class Main {
2    public static void main(String[] args) {
3      int a = 3; int b = 5;
4      int c = a * b;
5      System.out.println("縦幅3横幅5の長方形の面積は、" + c);
6    }
7  }
```

2行に分けても構いません

練習 1-3 の解答

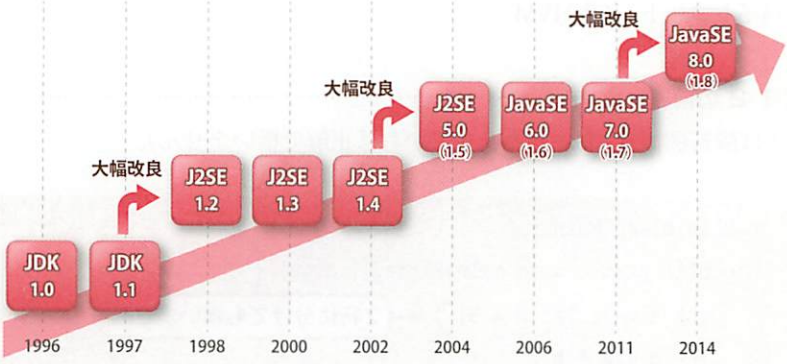
以下は解答例です(おおむね合っていれば正解で構いません)。

```
boolean result = true;
char favoriteCharacter = '駆';
double pi = 3.14;
long number = 314159265853979L;
String msg = "ミナトの攻撃！　敵に15ポイントのダメージを与えた。";
```

float pi = 3.14F でも可

Java の進化の歴史

Java が公開された 1996 年頃は C 言語などが主流で、Java はマイナーな言語の 1 つにすぎませんでした。その後、バージョンアップのたびに改良され、現在のように広く使われるようになりました。



① 3 度の大幅改良

Java の歴史の中では 3 度の「大幅改良」がありました。1 度目はバージョン 1.1 から 1.2 への改良です。1.2 は現在普及している Java の基礎を築いたバージョンです。2 度目はバージョン 1.4 から 1.5 への改良で、より利用しやすく敷居の低い言語にするために、文法の拡張や API の追加が行われました。そして 2014 年、ラムダ式等の機能を取り入れた 8.0 がリリースされました。

②変化したバージョン番号の振り方

1.4 から 1.5 へのバージョンアップが比較的大がかりであったため、公式のバージョン番号としては 1.5 ではなく 5 を使うことになりました。つまり、**バージョン 5 とバージョン 1.5 は同じものを指しています**。それ以降、6、7、そして 8 とバージョン番号は振られていますが、Java の内部では 1.7 や 1.8 という古いバージョン表記が利用されている部分もあります。