List 1[もっとも簡単なソースプログラム(Hello.java)]

```
public class Hello {
  public static void main( String args[] ) {
    System.out.println("hello!");
  }
}
```

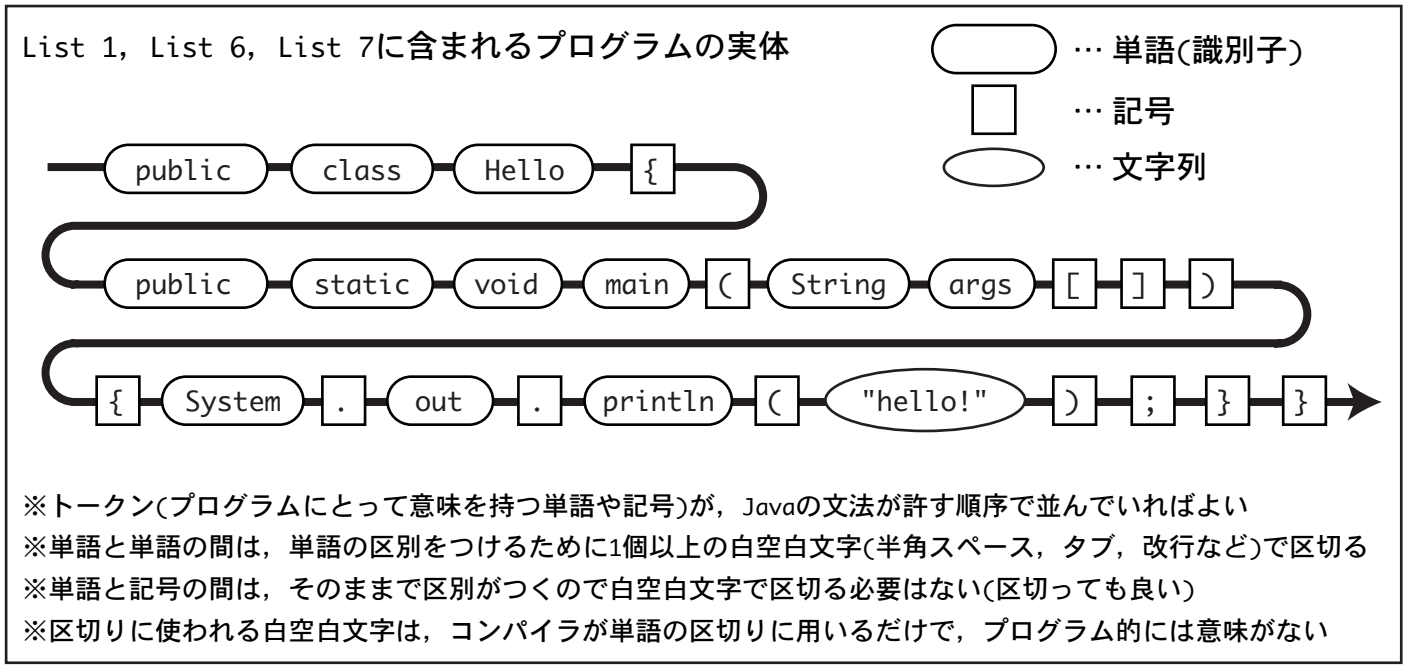
List 2[構造を把握しにくい書き方1(Hello.java)]

```
public class Hello {public static
void main(String args[]){ System.out.println("hello!");}}
```

List 3[構造を把握しにくい書き方2(Hello.java)]

```
public class Hello
{public static
void main(String
args[]){
System.  out. println
("hello!")
;}}
```

Fig. 1[List 1，List 2，List 3に含まれるプログラムの実体]



※トークン(プログラムにとって意味を持つ単語や記号)が，Javaの文法が許す順序で並んでいればよい
※単語と単語の間は，単語の区別をつけるために1個以上の白空白文字(半角スペース，タブ，改行など)で区切る
※単語と記号の間は，そのままで区別がつくので白空白文字で区切る必要はない(区切っても良い)
※区切りに使われる白空白文字は，コンパイラが単語の区切りに用いるだけで，プログラム的には意味がない

Fig. 2[プログラムに出てくる基本構造]

プログラムに出てくる基本構造

(1)Javaのプログラムによく出てくる基本構造

0個以上の単語や記号の列 { }

※ List 1などに含まれている例

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

※ 対応する{ }の間に，他の構造を入れることができる

(2)List 1, List 6, List 7に含まれる3重構造

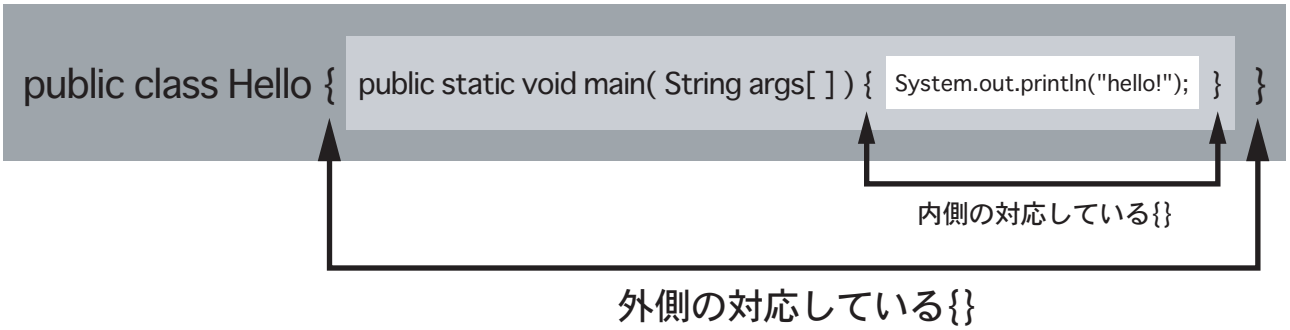


Fig. 3[List 1の書き方は構造が把握しやすい]

List 1の書き方は構造が把握しやすい

```
public class Hello {
    public static void main( String args[] ) {
        System.out.println("hello!");
    }
}
```

※List 1の書き方は，このように構造とそれが入れ子になっている状態を把握しやすい

Fig. 4[字下げの仕方]

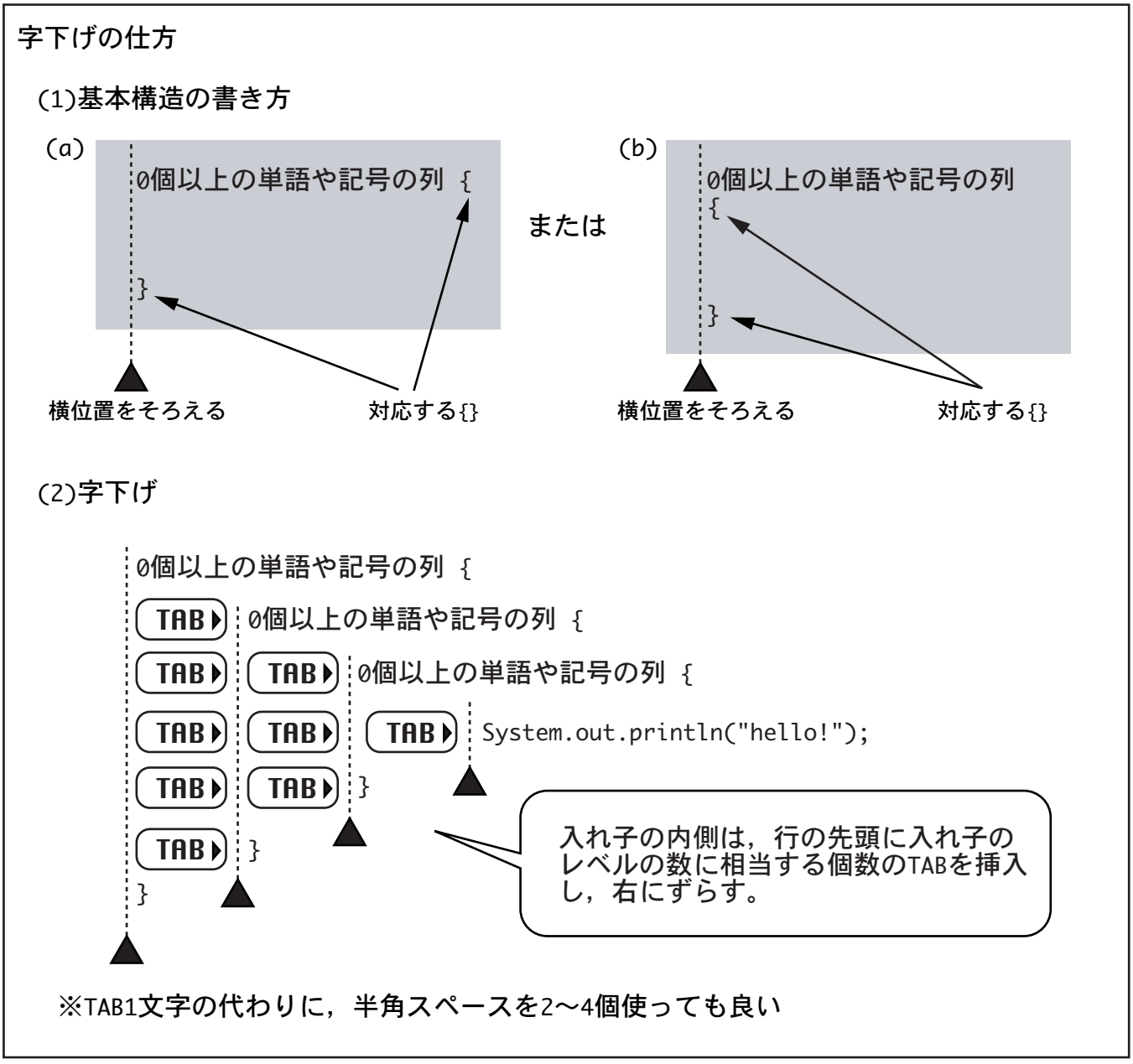


Fig. 5[対応するカッコを先に書く(List 1の例)]

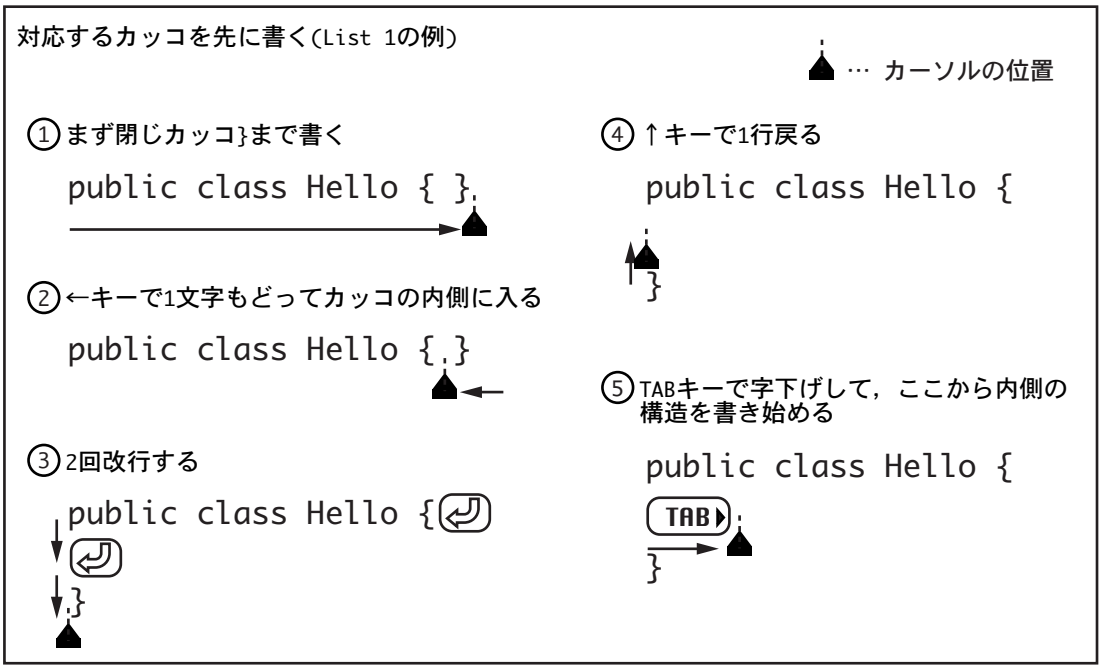


Fig. 6[構造がわかりやすいようにマーキングする]

構造がわかりやすいようにマーキングする

```
public class Hello {
    public static void main( String args[] ) {
        System.out.println("hello!");
    }
}
```

List 8[List 1にコメントを入れた例]

```
 /* 初めて自分で書いたプログラムです。
    画面に hello! と表示されます。      */

public class Hello {
  public static void main( String args[] ) {
    // ここで hello! と表示する
    System.out.println("hello!");
  }
} // class Hello
```

Fig. 7[コメントの書き方]

コメントの書き方

(1) 2重のスラッシュから行末まで　　(2) /* と */ の間

// ここにコメント内容を書く　　/* ここにコメント内容を書く */

※ //，/*，*/は2文字の記号をくっつけて書くこと。離して書いてはいけない。

※ (1)は1行の中だけに1つだけコメントを書ける。(2)は複数行にわたって1つのコメントを書ける。

※ (2)のコメントは入れ子になってはいけない(/* /* */ */ などはダメということ)

※ コンパイラはコメントを白空白として扱う

※ コメント内容には，日本語(全角文字)を使っても良い

Fig. 8[コメントアウトとコメント形式]

コメントアウトとコメント形式

(1)/* */形式コメントを含む部分を/* */形式でコメントアウトするとエラーになる

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

    System.out.println("hello!"); /* hello!と表示 */

  }
}
```

既に，/* */形式の入っている部分を/* */形式でコメントアウトしようとすると…

コメントアウト

```
public class Hello {
  public static void main( String args[] ) {
    /*
    System.out.println("hello!"); /* hello!と表示 */
    */
  }
}
```

/* */形式のコメントが入れ子になってしまうので，エラーになる！

(2)//形式だけのコメントを含む部分を/* */形式でコメントアウトしてもエラーにならない

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

    System.out.println("hello!"); // hello!と表示

  }
}
```

//形式のコメントなら，この部分を含む箇所を/* */形式でコメントアウトしても…

コメントアウト

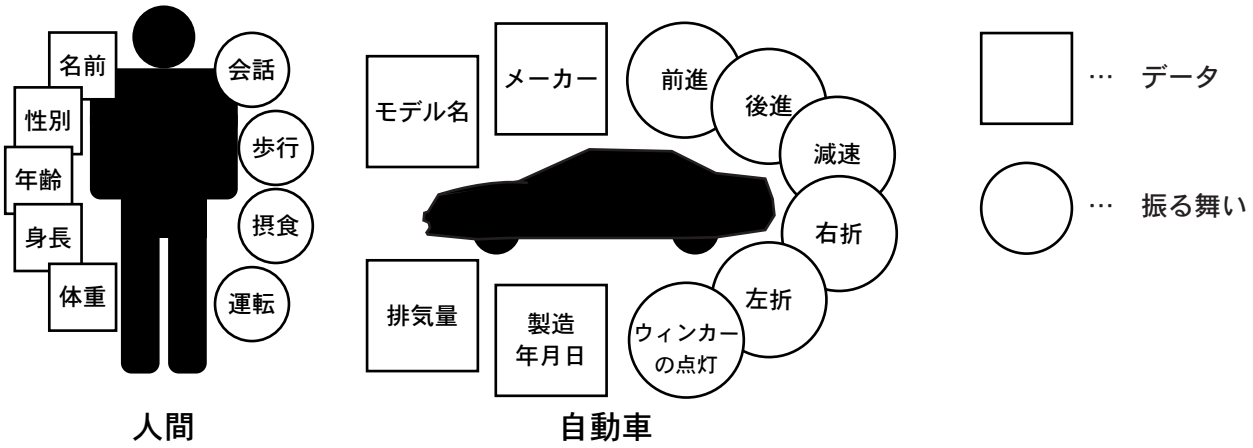
```
public class Hello {
  public static void main( String args[] ) {
    /*
    System.out.println("hello!"); // hello!と表示
    */
  }
}
```

/* */形式のコメントが入れ子になることは無いので大丈夫！

Fig. 9[クラス]

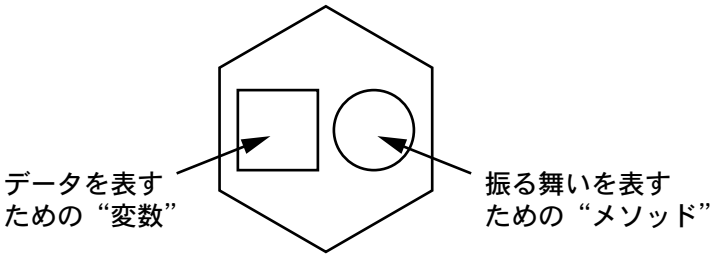
クラス

(1)事物を表現するには“データ”と“振る舞い”が必要



人間　　自動車

(2)クラスは“データ”と“振る舞い”をひとまとめにしたもの

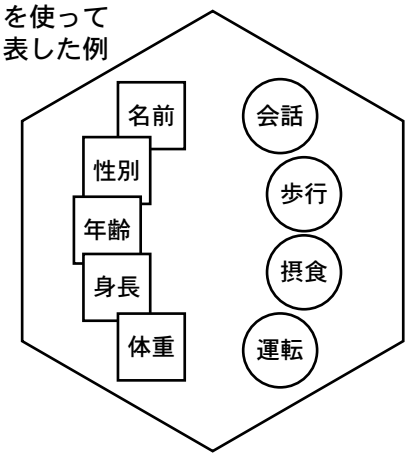


クラス

※ クラス内部にある変数やメソッドは，クラスの定義に含まれる“一員”という意味で，“メンバ”と呼ばれる

※ クラス自体，他のクラスのメンバとして使うことができる

(例) クラスを使って人間を表した例



クラスHuman

(3)クラスのメンバを指定する書き方

あるクラスのメンバを指定するには次のように書く

クラス名.メンバ名

ピリオド(.)でクラス名とメンバ名をつなげる

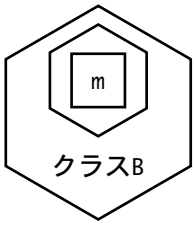
(例1) クラスAのメンバm

A.m

(例2) クラスAのメンバであるクラスBのメンバm

A.B.m

例2のイメージ



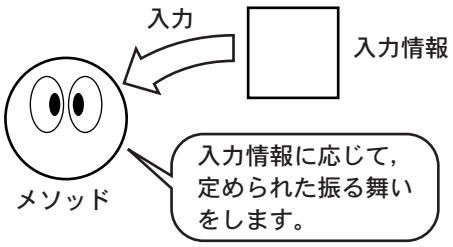
クラスA

Fig. 10[メソッドの概要]

メソッドの概要

(1)メソッド定義のおおまかな形

```
単語の列　メソッド名( 入力情報 ){
  振る舞いの具体的な内容
}
```



※メソッドは，処理の結果の情報を出力情報として返すこともできるが，
ここでは省略する

(2)あるクラスのメンバであるメソッドの基本的な呼び出し方

```
クラス名.メソッド名( メソッドに渡す具体的な入力情報 );
```

(3)List 1の2〜4行目のmainというメソッドの定義

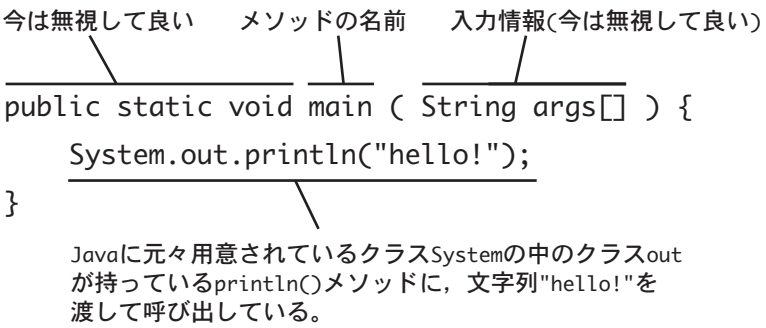
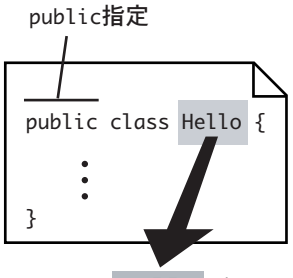


Fig. 11[Javaの約束事]

Javaの約束事

(1)

ソースプログラムのファイル名は，そのソースプログラムの中にあるpublic指定されたクラスと同じ名前に拡張子".java"をつけたものでなければならない



ファイル名: Hello.java

(2)

Java仮想マシンは，指定されたクラスファイル中のpublicなクラスが

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

という形式のmianメソッドを持っている場合，そのmainメソッドを呼び出す。これによって，プログラムの実行が開始される。

Fig. 1[ビットとバイト]

ビットとバイト

(1)ビット

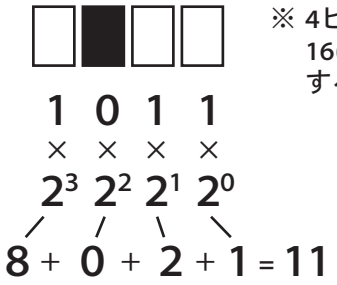
コンピュータでは，あらゆる情報はONかOFFかの2種類の状態を持つビット(bit)で表される

□ … ON　　■ … OFF

(2)複数のビットで整数値を表す

整数値も1ビットを一桁とする2進数で表現される
(このとき，ONを1，OFFを0として扱う)

(例)4ビットで表現された整数値11



※ 4ビットだと，0～15の16(=2^4)個の数値を表現することができる

(3)バイト

8ビットの固まりをバイト(byte)と呼ぶ。1バイトで0以上の整数を表すと，0～255の計256(=2^8)個の整数を表現することができる

(例)1バイトで表現された整数値171

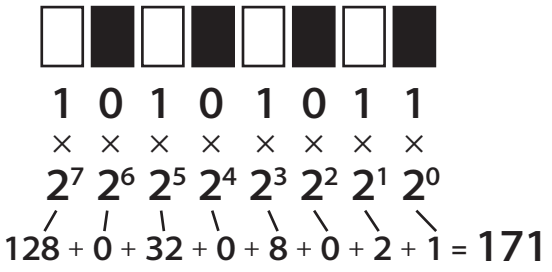
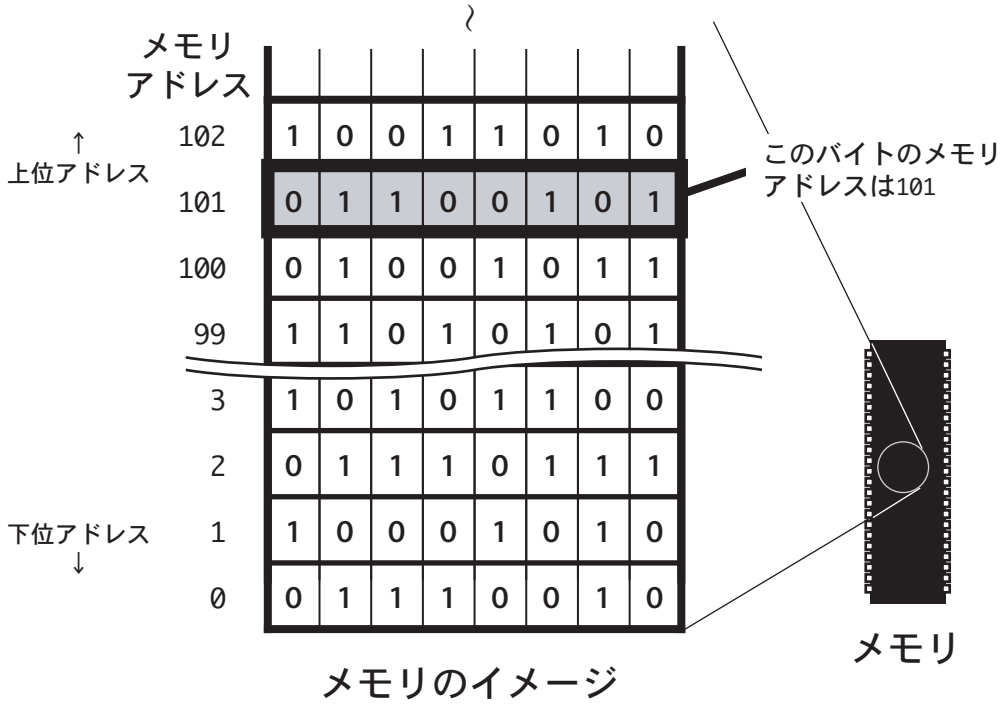


Fig. 2[メモリ上の情報表現]

メモリ上の情報表現



メモリのイメージ

※ メモリは，各バイトごとに通し番号による番地が割り当てられている。これをメモリアドレスと呼ぶ。CPUは，このメモリアドレスによってメモリ上にある情報にアクセスする

※ メモリアドレスの値が小さくなる方向を下位アドレス方向，大きくなる方向を上位アドレス方向と呼ぶ

※ コンピュータ上で表される情報は，メモリ上に必要なバイト数の領域が確保され，その領域のビットパターンで表現・保持される

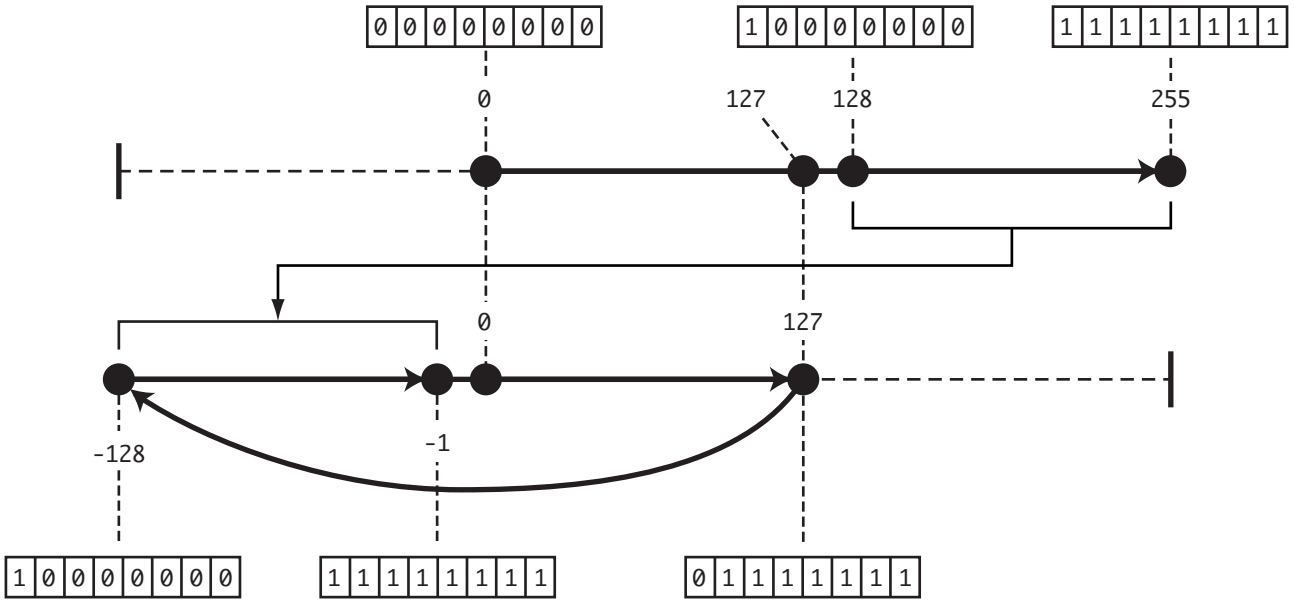
※ Javaでは，メモリアドレスを直接扱うことはない。しかしプログラミングをする上で，上図のようなメモリの仕組みと，メモリ上でどのように情報が表現・保持されているかを思い浮かべることができることは重要

Fig. 3[負の整数の表し方]

負の整数の表し方

負数も含めた整数を表すには，0以上の整数を表す場合(下図上段)の上半分を，負数の最小値から-1までの範囲を表すために使用する(下図下段)

(例)1バイト整数の場合



※負の整数を表す方法はいくつかあるが，この方法は「2の補数記数法」と呼ばれ，Java仮想マシンだけでなく，ほとんどのパソコン用CPUで用いられている

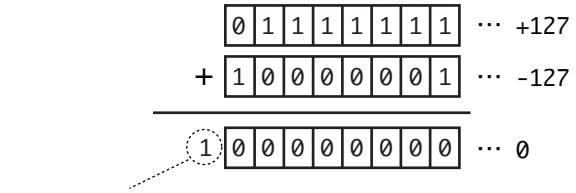
Fig. 4[2の補数記数法の特徴]

2の補数記数法の特徴

2の補数記数法は
- nビットなら-2^{n-1}～$2^{n-1}-1$の範囲の整数を表現する
- 一番左のビットが，負数の場合は常に1，正数の場合は常に0
- -1は，すべてのビットが1
- 引き算が足し算の手続きで実行可能(下図参照)

といった特徴も持つ

(例)1バイト2の補数記数法で，127-127を計算



繰り上がってあふれた1は無視する

※n-mを計算するとき，nと-mの2の補数記数法表現をビットごとに加算して，あふれた1を無視すればよい。

※引き算と足し算が，ビットごとの加算という同じ手続きで行えるので，加算用の回路だけを用意してやればよい

Fig. 5[文字コード]

文字コード

文字コードとは，文字を数値で表したもの。

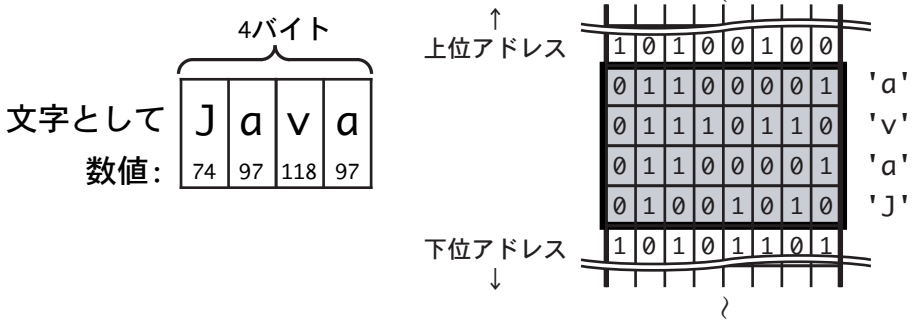
(1)ASCIIコード

文字	整数値
スペースや改行などの特殊文字	0 ～ 32
!"#$%&'()*+,-./	33 ～ 47
0 から 9	48 ～ 57
: ; < = > ? @	58 ～ 64

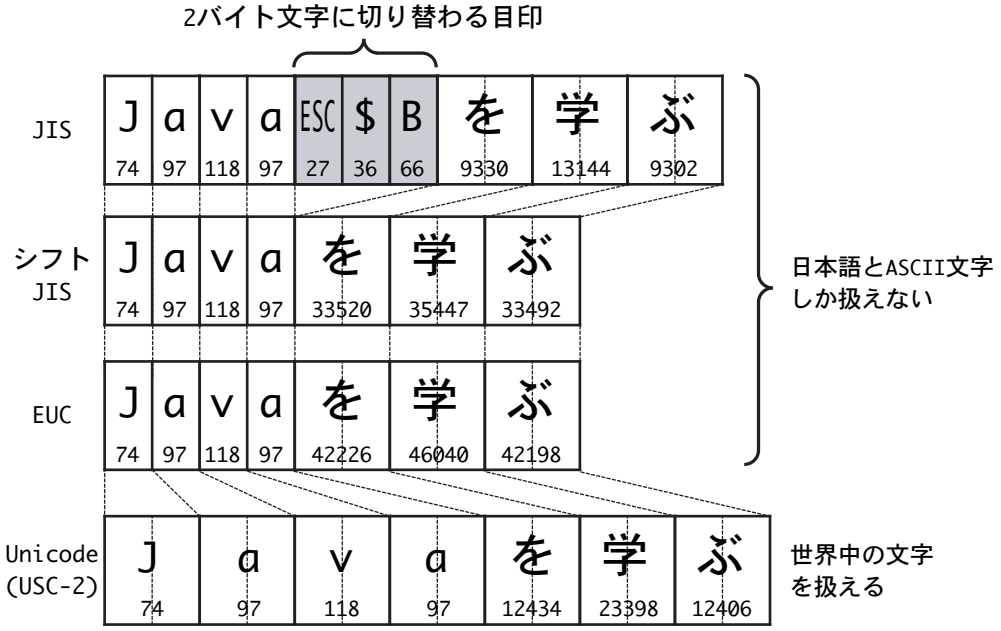
文字	整数値
A から Z	65 ～ 90
[\ ] ^ _ `	91 ～ 96
a から z	97 ～ 122
{ | } ~	123 ～ 126
DEL	127

※最後のDELは，キーボードのdeleteキーを押すと入力される特殊文字
※0～127の数値で128種類の文字を表すので，1文字を7bitで表現できるが，通常は1バイト(そのうち1ビットは未使用)で1文字を表す

(例) "Java"という文字列を数値としてみると…



(2)様々な文字コード



※JISの例の5文字目のESCは，キーボードのエスケープキーを押したら入力される特殊文字
※国際化したプログラムを書きやすいように，JavaはUnicodeを使用している

Fig. 6[浮動小数点数]

浮動小数点数

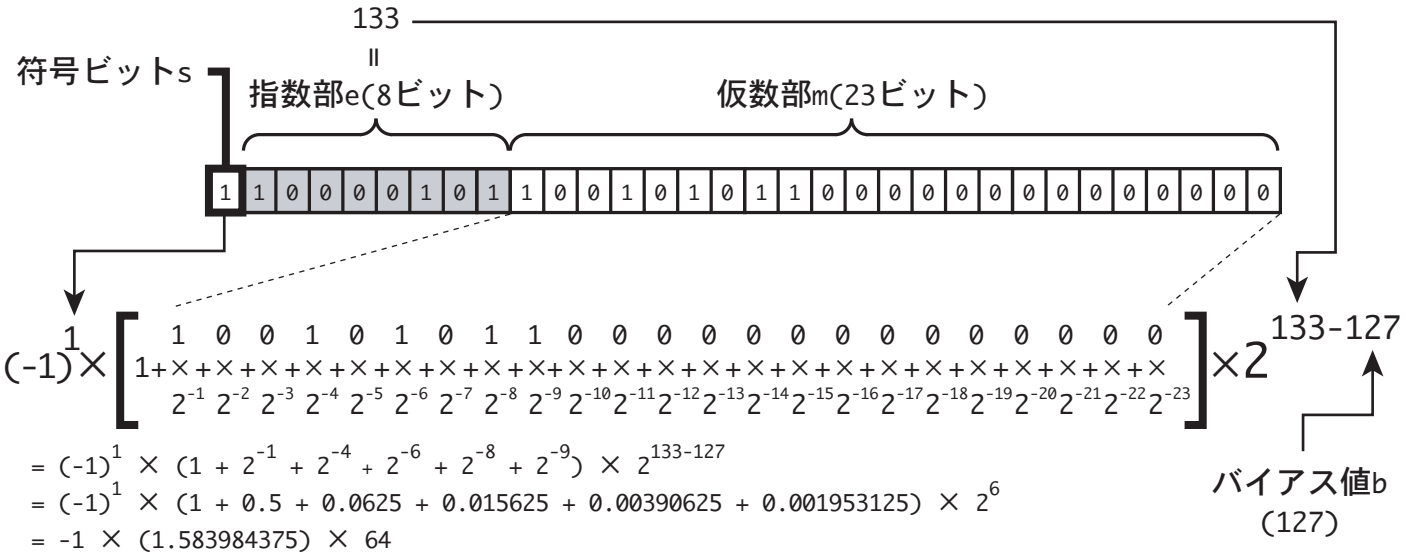
実数は，浮動小数点数という表現方法で表される。具体的には

$$(-1)^s \times 1.m \times 2^{(e-b)}$$

s … 符号ビット　　e … 指数部
m … 仮数部　　b … バイアス値

というような形で実数を表現し，s，m，eをそれぞれ2進数であらわす。bは形式によってあらかじめ定まっている。

(例)32ビット浮動小数点数



$= (-1)^1 \times (1 + 2^{-1} + 2^{-4} + 2^{-6} + 2^{-8} + 2^{-9}) \times 2^{133-127}$
$= (-1)^1 \times (1 + 0.5 + 0.0625 + 0.015625 + 0.00390625 + 0.001953125) \times 2^6$
$= -1 \times (1.583984375) \times 64$
$= -101.375$

Fig. 7[データの型]

データの型

(1)データの型とは

コンピュータで情報(データ)を表現・保持するには，整数や実数の種類を指定しなければならない。このデータの種類を“型(type)”と言う。

(2)Javaで扱う基本的なデータ型

(a)整数型 … 整数を表すのに用いられる

byte, short, int, long, char

(b)浮動小数点数型 … 実数を表すのに用いられる

float, double

Table 1[Javaで扱う整数型]

型名	バイト数	表現する範囲
byte	1	-128 ～ 128
short	2	-32768 ～ 32767
int	4	-2147483648 ～ 2147483647
long	8	-9223372036854775808 ～ 9223372036854775807
char	2	0 ～ 65535

※char型だけは符号なし整数で，Unicode文字1文字を表現するのに用いられる

Table 2[Javaで扱う浮動小数点数型]

型名	バイト数	指数部	仮数部	バイアス値	表現する範囲	表現可能な最小の絶対値
float	4	8ビット	23ビット	127	約3.4028235×10^{38} ～ 約-3.4028235×10^{38}	約1.4×10^{-45}
double	8	11ビット	52ビット	1023	約$1.7976931348623157\times10^{308}$ ～ 約$1.7976931348623157\times10^{308}$	約4.9×10^{-324}

※floatは単精度浮動小数点数，doubleは倍精度浮動小数点数と呼ばれる

※floatとdoubleの表現できる正確な最大の正数値は，それぞれFloat.MAX_VALUE，Double.MAX_VALUEで参照可能

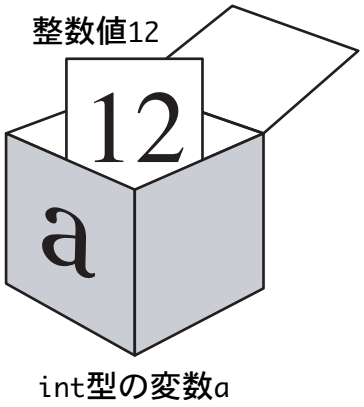
※floatとdoubleの表現できる正確な最小の絶対値は，それぞれFloat.MIN_VALUE，Double.MIN_VALUEで参照可能

Fig. 8[変数の概要]

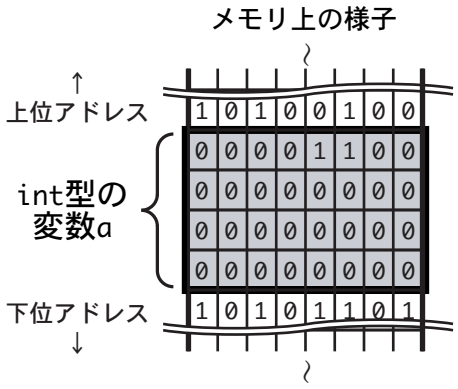
変数の概要

(1)変数は値を入れて記憶しておける名前つきの箱

(a)箱としてのイメージ　　(b)メモリ上の変数



int型の変数a



※ int型なので4バイトを占めていることに注意

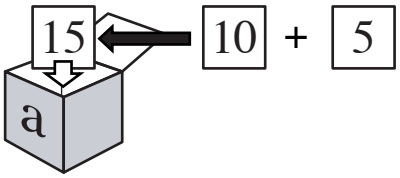
(2)変数に値を入れるには，代入演算子=を使う

(例1)int型変数aに計算 10+5 の結果値を入れる

```
a = 10 + 5;
```

このようにソースプログラムに直接書かれた値をリテラルと言う

ひとつの独立した式を書く場合，最後にセミコロン(;)を書く

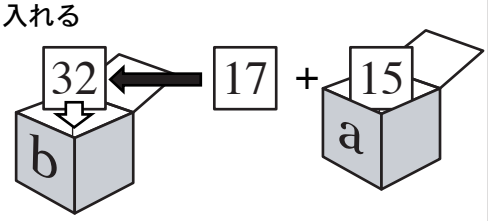


※ この結果，変数aには整数値15が入る

(例2)int型変数bに計算 17+a の結果値を入れる

```
b = 17 + a;
```

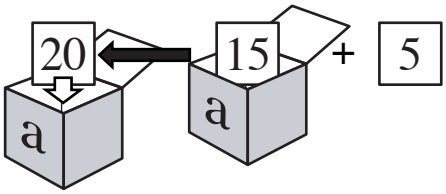
このように式の中で，変数を使うことができる。この場合，変数aの値が整数値15とすると17+15が計算され，その結果である32が変数bに代入される



※ この結果，変数bには整数値32が入る

(例3)int型変数aの値を5だけ増やす

```
a = a + 5;
```



※ この結果，変数aの値は整数値20に更新される

Fig. 9[基本的な変数の宣言の仕方]

基本的な変数の宣言の仕方

(1)基本形

```
変数の型名　変数名;
```

(例)

```
int a;
double d;
```

※メソッドの中で宣言された変数には，最初は未知の値が格納されている

(2)初期化を伴う宣言

```
変数の型名　変数名 = 初期値;
```

(例)

```
int a = 100;
double d = 1.23;
```

(3)同じ型の変数を1度に複数個宣言する

```
変数の型名　変数名1，変数名2，... ，変数名n;
```

※各変数について，初期化も行うことができる(下例参照)

(例)

```
int a, b, c = 5, d, e = 10;  // cとeだけ初期化をともなって宣言
double d1, d2, d3 = 0.3, d4; // d3だけ初期化をともなって宣言
```

Fig. 10[変数名の命名規則]

変数名の命名規則

変数名には，アルファベット(A〜Z，a〜z)，アンダーバー(_)，ドル記号($)，数字(0〜9)を使ってよい。変数名は何文字でも良い。ただし，最初の1文字には数字は使ってはならない。また，Javaのプログラムで特別な意味を持つキーワードという単語(Fig.11参照)と同じ名前にしてはならない。

※ 実際には，Unicodeで表される大部分の文字を変数名に用いることができるが，ここでは上記の文字に使用を限ることとする。なお，ドル記号($)の使用は今は使用しないことが推奨されている。

※ 慣習的に，変数名の最初の1文字はアルファベットの小文字にする

(a)正しい変数名の例

```
a, abc, a123, a_123, mySalary, distanceFromTokyoToKyoto
```

(b)間違った変数名の例

```
123a, abc
```

Fig. 11[Javaのキーワード]

Javaのキーワード

abstract	boolean	break	byte	case	catch	char
class	const	sontinue	default	do	double	else
extends	final	finaly	float	for	goto	if
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	

List 1[変数を使った例1(VarSample1.java)]

```
public class VarSample1 {
  public static void main (String args[]) {

    int a = 11;    // int型変数aを宣言して11で初期化
    int b =  3;    // int型変数bを宣言して3で初期化

    System.out.println( "-----------" );
    System.out.println( a );        // int型変数aの値を表示
    System.out.println( b );        // int型変数bの値を表示

    System.out.println( "-----------" );
    System.out.println( a + b );    // int型整数どうしのたし算
    System.out.println( a - b );    // int型整数どうしのひき算
    System.out.println( a * b );    // int型整数どうしのかけ算
    System.out.println( a / b );    // int型整数どうしの割り算
    System.out.println( a % b );    // int型整数どうしの剰余算

    System.out.println( "-----------" );
    System.out.println(  a + b  * 10 );   // "41"  と表示
    System.out.println( (a + b) * 10 );   // "140"  と表示

    int c1, c2, c3, c4, c5; // int型変数c1～c5を宣言
    c1 = a + b;    // int型整数どうしのたし算
    c2 = a - b;    // int型整数どうしのひき算
    c3 = a * b;    // int型整数どうしのかけ算
    c4 = a / b;    // int型整数どうしの割り算
    c5 = a % b;    // int型整数どうしの剰余算

    System.out.println( "-----------" );
    System.out.println( c1 );
    System.out.println( c2 );
    System.out.println( c3 );
    System.out.println( c4 );
    System.out.println( c5 );

  }
}
```

List 2[List1の実行結果]

```
-----------
11
3
-----------
14
8
33
3
2
-----------
41
140
-----------
14
8
33
3
2
```

Fig. 12[println()による数値の表示]

println()による数値の表示

prinln()メソッドに数値を渡すと，その数値を表す文字列に変換されて
println()メソッドに渡される。

(例)println()メソッドにint型変数a(値は11)を渡した場合

```
System.out.println(  a  );
                      ↓      … 変数aの値を表す文字列"11"に変換される
System.out.println( "11" );
```

List 3[変数を使った例2(VarSample2.java)]

```
public class VarSample2 {
  public static void main (String args[]) {

    double a = 11.0;    // double型変数aを宣言して11.0で初期化
    double b =  3.0;    // double型変数bを宣言して3.0で初期化

    System.out.println( "-----------" );
    System.out.println( a );        // double型変数aの値を表示
    System.out.println( b );        // double型変数bの値を表示

    System.out.println( "-----------" );
    System.out.println( a + b );    // double型浮動小数点数どうしのたし算
    System.out.println( a - b );    // double型浮動小数点数どうしのひき算
    System.out.println( a * b );    // double型浮動小数点数どうしのかけ算
    System.out.println( a / b );    // double型浮動小数点数どうしの割り算
    System.out.println( a % b );    // double型浮動小数点数どうしの剰余算

    System.out.println( "-----------" );
    System.out.println(  a + b  * 10 );   // "41.0"  と表示
    System.out.println( (a + b) * 10 );   // "140.0"  と表示

  }
}
```

List 4[List3の実行結果]

```
-----------
11.0
3.0
-----------
14.0
8.0
33.0
3.6666666666666665
2.0
-----------
41.0
140.0
```