第0章　C/Java言語の共通事項摘要

テキスト本体では、C 言語の知識を前提として Java 言語を説明する形式をとっている。そのため、この章では、テキスト本体で仮定する C 言語の知識（C 言語と Java 言語に共通する事柄）をダイジェストで紹介する。

0.1　全体的なこと

コンパイル　とは、ソースファイル（人間が読む･書く形式）を実行ファイル（CPU が直接理解できる形式）などに、翻訳することである。

注釈（コメント）　とは、ソースファイル中の人間向けのメッセージで、コンパイラは無視する部分である。C 言語では /*（空欄 0.1.1）から*/（空欄 0.1.2）までが注釈である。さらに新しい C 言語の仕様（C99）では//（空欄 0.1.3）から行末までという形も利用できる。（Java 言語でもどちらのコメントの形式も利用できる。）

`int`　とは、整数を扱うための型である。

`double`　とは、いわゆる実数（正確には浮動小数点数）を扱うための型である。もちろん、実数と言っても精度には限界がある。

キーワード　`if` や `else` などプログラミング言語にとって特別な意味のある単語をキーワード（keyword）と呼ぶ。変数名などに使用することはできない。（ただし、変数名の一部に使用するのは構わない。）

自由形式　C 言語や Java 言語では、原則としてレイアウト（空白の数や改行）はプログラムの意味に影響を及ぼさない。空白がいくつ連続しても空白 1 文字と同じであり、改行も空白と同じである。

0.2　式と演算子

文字列リテラル　とは、複数の文字を二重引用符（"）で囲んだものであり、文字列のデータを表す。

拡張表記　¥n は改行、¥t はタブ文字を表す。このような、¥を使った書き方を拡張表記という。

```
System.out.printf("abc\n12\tcde\n34\t56\tfgh\n");
```

文字リテラル　1 つの文字を一重引用符（'）で囲んだもののことである。¥n や ¥t などの拡張表記は 1 文字として扱われる。

printf の変換指定のまとめ　printf（Java では System.out.printf）の第 1 引数のなかで、%から始まる部分は変換指定と言い、第 2 引数以降の値に順に置き換えられる。整数（10 進数）は %d(空欄 0.2.1)、浮動小数点数は%f(空欄 0.2.2)、文字列は%s(空欄 0.2.3) を使う。

```
System.out.printf("半径␣%d␣の円の周長は␣%f\n", r, 2*r*3.14);
```

高度な変換指定　以下のような変換指定は必要に応じて調べれば良い。

説明	例	出力
桁数揃え	System.out.printf("[%3d]", 1)	[1]
桁数揃え（先頭を 0）	System.out.printf("[%03d]", 1)	[001]
小数点以下の桁数指定	System.out.printf("[%.3f]", 1.0/3)	[0.333]
16 進数で表示 (小文字)	System.out.printf("[%x]", 127)	[7f]
16 進数で表示 (大文字)	System.out.printf("[%X]", 127)	[7F]

変数　とは、数値などのデータを収納するための「箱」である。C 言語や Java 言語では、変数を使うためには事前に型を指定する宣言が必要である。

```
int vx;       /* int（整数）型の変数 vx の宣言 */
double fx;    /* double（倍精度浮動小数点数）型の変数 fx の宣言 */
int vx, vy;   /* int（整数）型の変数 vx と vy の宣言 */
```

演算子　（オペレーター）とは、+, -, *, /のように 演算の働きを持った記号のことである。

オペランドとは、その演算の対象となる式のことである。

代入演算子　代入とは、変数の値を書き換えることである。「変数 = 式」という形式で、右辺の式の値を左辺の変数に代入する。

代入（「変数 = 式 」という形式）も式であり、値（代入された値と同じ）を持つ。代入演算子（=）は右結合(空欄 0.2.4)である。つまり、x = y = 0 は x = (y = 0)(空欄 0.2.5) と解釈される。

/ 演算子、% 演算子

整数 / 整数

という演算では、整数としての割算（小数点以下は切捨て）としての結果が得られる。

整数 % 整数

では、余りを求める。

```
System.out.printf("%d␣/␣%d␣=␣%d\n", 5, 3, 5/3);
System.out.printf("%d␣%␣%d␣=␣%d\n", 8, 5, 8%5);
```

等価演算子 ==は両辺の値が等しければ真(空欄 0.2.6)を、等しくなければ偽(空欄 0.2.7)を返す演算子である。==と逆に等しくないかどうかを判定する演算子は!=(空欄 0.2.8)である。

関係演算子 以下の4つがある。

<	左辺が右辺よりも小さい
>	左辺が右辺よりも大きい
<=	左辺が右辺よりも小さいか等しい
>=	左辺が右辺よりも大きいか等しい

論理演算子 は以下のような演算子である。左右非対称である — つまり左辺を評価して値が決まれば、右辺は評価しない — ことに注意する。

演算子	呼び方	説明
&&	論理 AND 演算子	左辺を評価して、偽であれば、偽を返す。 真であれば、右辺を評価してその値を返す。
||	論理 OR 演算子	左辺を評価して、真であれば 真を返す。 偽であれば、右辺を評価してその値を返す。

複合代入演算子 *=, /=, %=, +=, -=, などのことである。例えば、 `sum += no` は `sum = sum + no`(空欄 0.2.9)と同じ意味になる。

後置増分演算子・前置増分演算子

a++	a の値を一つだけ増やす	（式全体の値は、増加前の変数の値）
a--	a の値を一つだけ減らす	（式全体の値は、減少前の変数の値）
++a	a の値を一つだけ増やす	（式全体の値は、増加後の変数の値）
--a	a の値を一つだけ減らす	（式全体の値は、減少後の変数の値）

キャスト（cast）　とは、明示的に型変換することである。

　(型) 式

という形で、「式」の値を「型」としての値に変換する。例えば、

```
int na, nb
...
(double)(na + nb) / 2
```

では、double 型としての割算が行われる。（演算子の優先順位に注意する。割算よりもキャストが先に行われる。）

```
System.out.printf("%d␣/␣%d␣=␣%f\n", 5, 3, (double)5/3);
```

0.3　文と宣言

式文

　式 ;

“式” (expression)（通常、代入や入出力など、副作用（空欄 0.3.1）（式の値以外の作用）を持つ式）の後にセミコロン (;) をつけたものは、“文” (statement) になる。

　この形の文を式文という。

```
x = 2;
System.out.printf("x␣=␣%d\n", x);
```

複合文（ブロック）　文の並びを波括弧（ブレース — {と}（空欄 0.3.2） —）で囲んだものを複合文またはブロックという。（C では文の前（Java では文のあいだも可）にいくつかの宣言があってもよい。）複合文は文法上単一の文と見なされる。

　複合文中の文は上（同じ行内なら左）から順に一つずつ実行される。

複合文（ブロック）内での宣言　ブロックの中で宣言された変数は、そのブロックでのみ有効である。

初期化子　変数は次の形で初期化することができる。

　型名　変数名$_1$ = 初期化子$_1$, ...　, 変数名$_n$ = 初期化子$_n$;

初期化子（initializer）は今のところ “式”（expression）と思っておいて良い。

if 文

　if (式) 文

式を評価して、その値が真であれば、文を実行する。式の値が偽であるときは、何もしない

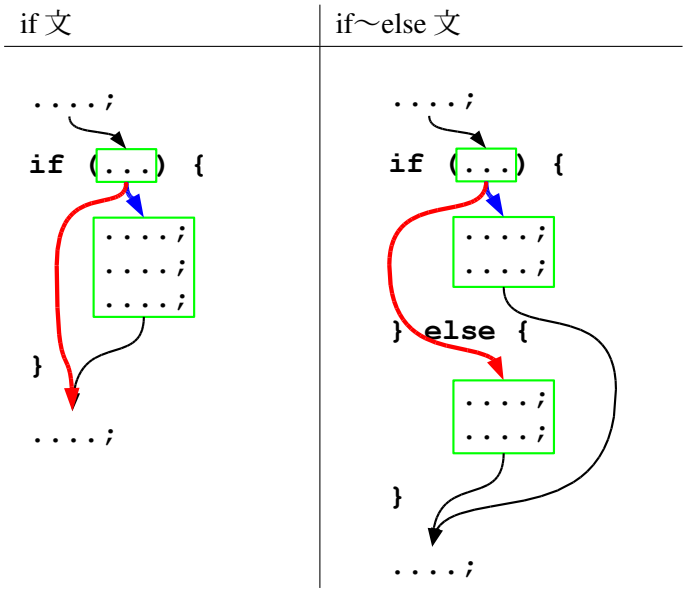
```
if (score >= 60) {
  System.out.printf("合格");
}
```

if〜else 文

```
if ( 式 ) 文1 else 文2
```

式を評価して、その値が真であれば、文 $_1$ を実行する。式の値が偽であるときは、文 $_2$ を実行する。

```
if (score >= 60) {
  System.out.printf("合格");
} else {
  System.out.printf("不合格");
}
```



入れ子になった if 文

```
if (no == 0) {
    System.out.println("その数は␣0です。");
} else if (no > 0) {
    System.out.println("その数は正です。");
} else {
    System.out.println("その数は負です。");
}
```

これは、単に else の次の文が、また if 文になっているだけのことである。

switch 文 ある式の値（基本的には整数型）によって、プログラムの流れを複数に分岐するときに使う。

```
switch ( 式 ) 文
```

switch 文は式を評価して、case と :の間に書かれた値と一致するところにジャンプする。(どの case にも一致しないときは、default : にジャンプする。) ただし、break 文に出会うと、一気に switch 文を飛び出る。

case〜:や default:のようにプログラムの飛び先を示す目印をラベル（名札）と呼ぶ。

while 文

```
while ( 式 ) 文
```

式が偽でない限り、文（ループ本体）を繰り返し実行する。

注: ループ本体が一度も実行されないことがある。

for 文

```
for ( 式1; 式2; 式3 ) 文
```

ループに入る前にまず式 $_1$ を実行する。

式 $_2$ が真である間、文，式 $_3$ を繰返し実行する。

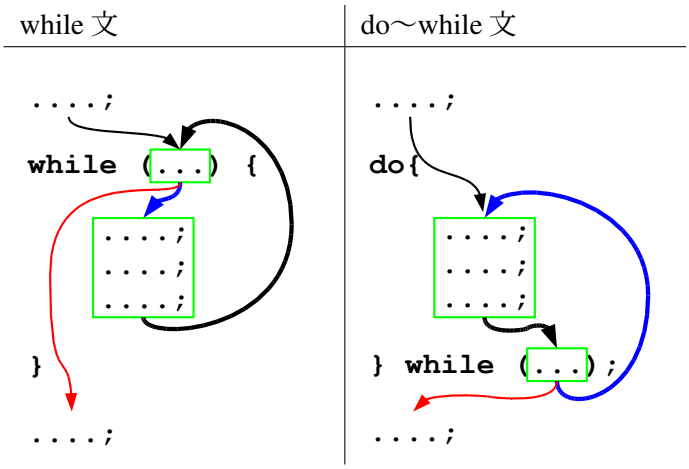
```
for (i=0; i<5; i++) {
  System.out.printf("%d␣*␣%d␣=␣%d\n", i, i, i*i);
}
```

do〜while 文

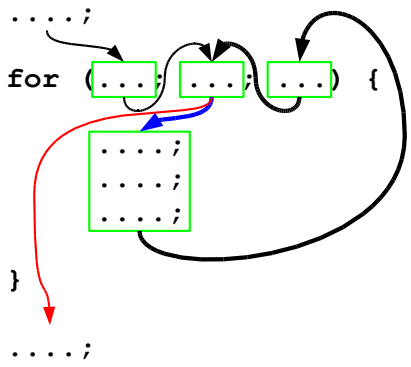
```
do 文 while ( 式 ) ;
```

まず、文（ループ本体）を実行する。式が偽にならない限り、文の実行を繰り返す。

注: 必ず 1 回はループ本体を実行する。



for 文



多重ループ for 文や while 文などのループ本体が、また for 文や while 文などの繰返し文になっていることを二重ループという。二重・三重・… ループをまとめて、多重ループという。

配列 同一の型のデータを集めて、番号（添字）でアクセスできるようにしたもの。C 言語や Java 言語の配列の添字は0(空欄 0.3.3) から始まる。

break 文

```
break ;
break ラベル ;        // Java のみ
```

（もっとも内側の）繰返し文（while 文, for 文, do〜while 文）を抜け出る。（外側の繰返し文を一気に抜け出るには、C 言語では goto 文、 Java 言語ではラベルつきの break 文を用いる。）

continue 文

```
continue ;
continue ラベル ;    // Java のみ
```

（もっとも内側の）繰返し文のはじめに戻る。（while 文, do〜while 文の場合は条件式、 for 文の場合は第 3 式に戻る。）

0.4 関数

関数とは 繰返し使うプログラムの一部の命令列を部品として、再利用できるようにしたもの。

関数定義とは

分類	一般形
関数定義	**型** 関数名 (**型** 変数名 , . . . , **型** 変数名) 複合文

補足: 関数定義には型が必要(空欄 0.4.1)
かっこの中の変数は仮引数(空欄 0.4.2)（parameter）と呼ばれる。

関数呼出し式とは

分類	一般形
関数呼出し式	関数名 (式 , . . . , 式)

補足: 関数呼出しには型は不要(空欄 0.4.3)
かっこの中の式は実引数(空欄 0.4.4)（argument）と呼ばれる。
　これで文法上、式（expression）になる。
　関数を呼出すと、プログラムの実行は呼び出された関数の定義の先頭に移り、実引数の値が仮引数の変数の初期値になる。

return 文

```
return 式 ;
return ;
```

関数の呼出し元に値を返す。つまり、プログラムの実行が関数の呼出し元に戻り、return 文の式の値が、関数呼出し式の値になる。式のない return 文は値を返さない関数のときに使用する。
　関数本体の最後の } にたどり着いたときも、プログラムの実行は関数の呼出し元に戻る。

値呼び（call by value）　引数は基本的に値がやりとりされる。関数呼出しのたびに仮引数のための新しいメモリ領域（“箱”）が用意される。仮引数の変数に代入を行なっても、呼出し元には影響を与えない。

値を返さない関数　戻り値型のところに`void`(空欄 0.4.5)と書く。

有効範囲（スコープ、scope）　変数には有効範囲がある。同じ変数名でも有効範囲が異なれば別の変数になる。

- ブロックの中で宣言された変数は、そのブロック（宣言された場所から、ブロックの最後まで）が有効範囲となる。
- 関数の仮引数は、その関数本体が有効範囲となる。

0.5 文法のまとめ

式（expression）とは これまでのところ、

分類	一般形	補足説明
変数		x, i など
整数リテラル		1, 0, 100, 0xff など
浮動小数点数リテラル		3.14, 2.0, 6.6626e-34 など
文字列リテラル	"複数個の文字"	"Hello¥n"など
文字リテラル	'1文字'	'a', '¥n' など
関数呼出し	関数名(式, ... , 式)	pritnf("Hello%d¥n", i) など
単項演算	単項演算子 式	単項演算子は +, -, ... など
後置演算	式 後置演算子	後置演算子は ++, --のみ
二項演算	式 二項演算子 式	二項演算子は +, -, *, /, = ... など
三項演算	式 ? 式 : 式	
かっこ	(式)	演算の順番を指定するため
キャスト	(型) 式	明示的な型変換

宣言（declaration）とは これまでのところ、

分類	一般形	補足説明
変数宣言	型 変数 = 式 , ... , 変数 = 式 ;	型は int, double など。

補足:「= 式」の部分は省略可能

文（statement）とは　これまでのところ、

分類	一般形	補足説明
式文	式 ;	通常、式は代入式など 副作用があるもの
return 文	return 式 ;	式の周りにかっこは必要なし
	return ;	値を返さない場合
if 文	if (式) 文	
if〜else 文	if (式) 文 else 文	
複合文（ブロック）	{ 宣言 ... 文 ... }	
switch 文	switch (式) 文	
ラベル付き文	case 整数リテラル : 文	
	default : 文	
break 文	break ;	
do〜while 文	do 文 while (式) ;	
while 文	while (式) 文	
for 文	for (式 ; 式 ; 式) 文	
continue 文	continue ;	
空文	;	“何もしない” 文、 {}と書いても同じ。