18 Java文法の初歩

18–1 コメント

- /* から */ までは、C言語の場合と同様にコメント。(人間向け)

  (このうち /** で始まるものについては、プログラムのドキュメントを生成する際に利用される。
  ドキュメントを生成するコマンドは javadoc 。
  ⟶第22.2節)

- // から その行の最後までは注釈。

補足(TODOコメント):

既存のプログラムを眺めていると、

```
//TODO ...
```

という風に ‘‘TODO’’ や ‘‘FIXME’’, ‘‘XXX’’ で始まるコメントに出会うことがある。これらの意味は次の通りである。

`//TODO` 作業 ... 作業 ということをやらなければならない。
(to do)

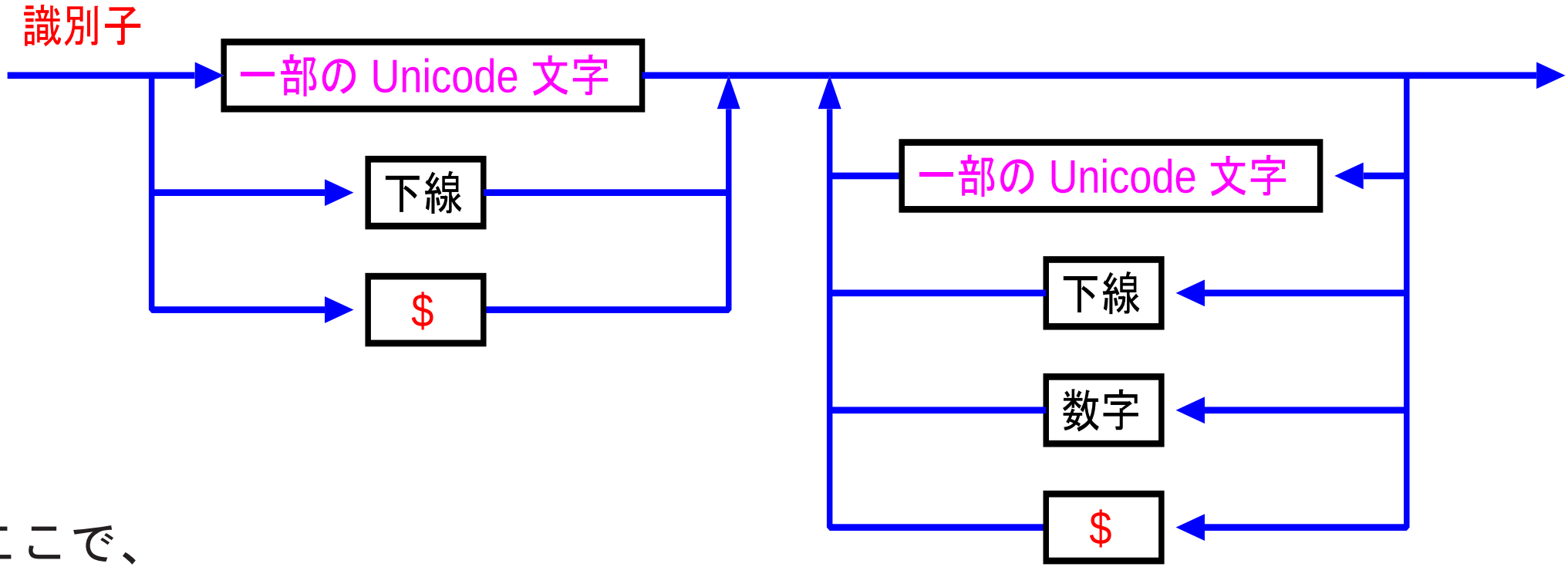
`//FIXME` 修正作業 ... 正しく動いてないので
修正作業 に記述されている様な修正が必要。
(fix me)

`//XXX` 間違い ... とりあえず動いてはいるが
間違い に記述されている様に正しくない。

18–2 識別子とキーワード

識別子：

変数等に付ける名前としては、次のものが許される。



ここで、

「一部のUnicode文字」の中には、ASCII英大文字，ASCII英小文字，日本語文字，... 等を含む。

キーワード, および定数として扱われる null，true，false は除く。

キーワード：

abstract	assert	boolean
break	byte	case
catch	char	class
const	continue	default
do	double	else
enum	extends	final
finally	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	

コーディング規約：

コードの可読性・保守性を高めたり不具合を招く可能性を抑制したりするために、コーディング規約がいくつか示されている。

（日経ソフトウェア編「ゼロから学ぶ！最新 Java プログラミング」p.156 表1，一部手直し）

名称	Web ページのアドレス
電通国際情報サービス版	http://www.objectclub.jp/community/codingstandard/JavaCodingStandard2004.pdf
オブジェクト倶楽部版	http://www.objectclub.jp/community/codingstandard/CodingStd.pdf
Java 言語コーディング規約（Sun コーディング規約）	http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html（原文），http://www.oracle.com/technetwork/java/codeconvtoc-136057.html（原文），http://numata.designed.jp/javacodeconv/（和訳）
頑健な Java プログラムの書き方	http://www.alles.or.jp/~torutk/oojava/codingStandard/writingrobustjavacode.html

これらのコーディング規約の中には 、
変数名に対する命名の仕方、書式、不具合を招くコードと回避例、等が示されている。
実際には、互いに矛盾した指針もある。
しかし、こういった規約にも十分留意すべきである。

例えば、
識別子に対する名前の付け方として 次の様な指針が一般的である。

◇ 英単語など、意味のある単語を並べて名前を付ける。

◇ 「意味のある単語」として、漢字等の特殊な文字は使わない。

◇ 「意味のある単語」としては、不可解な短縮形は使わない。

◇ ドル記号 ($) は基本的には使わない。

◇ アンダースコア (_ , 下線記号) は特別な場合以外は使わない。

18–3 基本データ型

整数型：

型名	ビット数	表現方式	暗黙の初期値
byte	8	signed & 負数は2の補数で	0
short	16	signed & 負数は2の補数で	0
int	32	signed & 負数は2の補数で	0
long	64	signed & 負数は2の補数で	0

浮動小数点数型：

型名	ビット数	表現方式	暗黙の初期値
float	32	IEEE規格754	0.0
double	64	IEEE規格754	0.0

文字型：

型名	ビット数	表現方式（文字コード）	暗黙の初期値	備考
char	16	Unicode（符号なし整数で文字を表す。）	\u0000（16進の'0000'）	算術演算可

論理型：

型名	ビット数	表現方式	暗黙の初期値	備考
boolean	1	0がfalse, 1がtrue	false	算術演算可

注意：

「暗黙の初期値」というものが定められてはいるが、
局所変数が自動的に初期設定される訳ではない。

18–4 リテラル(定数)

Javaでは定数のことをリテラルと呼んでいる。

整数型リテラル:

- 10進，16進，8進の表記が可能。(C言語と同様。)
- intで表せればint型，さもなければlong型。但し、最後に L または l を付けるとlong型になる。

浮動小数点数型リテラル:

- C言語の場合とほぼ同じ。(標準はdouble型で、.....　。)
- double型であることを明示したい場合は最後に D または d を付ける。

文字型リテラル：

- C言語の場合と同様、各文字に文字番号が割り当てられており、文字リテラルの値は割り当てられた文字番号になる。

- C言語の場合と同様、文字を1重引用符で囲んで表す。しかし、Javaは Unicode を採用しているので、(内部では) どの文字も16ビットで表される。

- Unicodeの値を16進表記(4桁)のエスケープシーケンス \u0000~ \uffff で表すことも出来る。例えば、'\u0041' は 'A' と同じ。(どちらも 65 という値を持つ。)

- 上位8ビットが全て0の文字を8進表記(3桁以下)のエスケープシーケンス '\0'~ '\7', '\00'~ '\77', '\000'~ '\377' で表すことも許されている。例えば、'\037' 。(C言語との互換性を維持するため。)

- C言語の場合と同様、特殊な文字はエスケープシーケンスによって表すことが出来る。

表記	意味	同等な16進表記	値
'\b'	後退(backspace)	'\u0008'	8
'\t'	タブ(tab)	'\u0009'	9
'\n'	改行(newline)	------	10
'\f'	紙送り(form feed)	'\u000C'	12
'\r'	復帰(return)	------	13
'\"'	2重引用符	'\u0022'	34
'\''	引用符	'\u0027'	39
'\\'	バックスラッシュ	'\u005c'	92

'\n'と'\u000A'が違う理由:

16進表記のエスケープシーケンスをUnicode文字に変換する作業がコンパイルの初期段階で行われ、直後にLF文字(改行)とCR文字(復帰)は行末記号(line terminator)に変換されるため、'\u000A'や'\u000D'は無効なものとなってしまう。 そういう意味で、'\n'と'\u000A'は違うし、'\r'と'\u000D'は違う。

論理型リテラル：

- `true` と `false` の2つ。

18–5 補足 文字コードの話

日本語コード体系には次の4つがあります。

- JIS漢字符号体系：
  日本語を扱うためには漢字が不可欠であるので、JISでは漢字符号(Kanji code)が1978年に制定され1983年,1990年に改訂されている 。JIS漢字符号は2バイト/16ビットのビット列で構成され、登録可能な漢字数2^{16}=65536の内第1水準漢字(平仮名，片仮名,英数字，特殊文字も含む)として2965字，第2水準漢字として3390字，補助漢字として5810字，計12156字が定められている。

- MS漢字コード体系：
  シフトJISコード系とも呼ばれるが、JISでなくJIS漢字符号体系の変形である。Escシーケンスを用いずに漢字符号と8ビット符号を混在させられる様に、1983年に米国Microsoft社,アスキー,日本IBM,三菱電機によって定義されたコード体系であり、パソコン/Windowsの世界では事実上の標準になっていた。

- EUC(Extended Unix Code)コード体系：
  JIS漢字符号体系の変形であり、UNIX-JISコード,またはUJISとも呼ばれている。日本語UNIXシステム諮問委員会の検討(AT&T社からの要請による,1984~ 5)の結果を基に1986年にAT&T社が定めたコード体系であり、長い間(2007年頃まで？) UNIX/Linuxにおける標準コードとして用いられていた。

- Unicode体系：
  世界各国の文字体系を共通化するために、IBM社, Sun Microsystems社, Microsoft社,ノベル社などの米国企業が中心になって設立したUnicコンソーシアムが提唱したもので、1993年にはISO (International Organization for Standardization)の標準に、1995年にはJIS規格となった。Unicodeの中にも幾つかの符号化方式があるが、その内UTF-8はASCIIに上位互換して最も一般的に利用されている。UTF-16 はWindowではXP以降の内部コードに使われている。

補足： 既に分かっていると思いますが、実習室では、JIS，シフトJIS，EUCの間のコード変換は次のように行ないます。

任意→EUC ··· nkf -e [テキストファイル名]
任意→JIS ··· nkf -j [テキストファイル名]
任意→シフトJIS ··· nkf -s [テキストファイル名]
任意→UTF-8 ··· nkf -w [テキストファイル名]
任意→UTF-16 ··· nkf -w16 [テキストファイル名]
（UTF-16がBig Endian,BOMなしの場合。）
指定→指定 ··· iconv -f [変換前コード] -t [変換後コード]
定可能な文字コードを知りたければ、 iconv -l とする。）

いずれの場合も、変換結果は標準出力に出されるので、変換結果をファイルに出力するためには、リダイレクションを使って例えば

nkf -w [テキストファイル名] > [UTF-8ファイル名]

とします。また、実習室でテキストファイルをプリンタに出力するためには、パイプを使って例えば

a2psj [テキストファイル名] | lpr -P [プリンタ名]

18–6 変数の宣言

- C言語と同様の書き方。
- 局所変数の場合は、C言語の場合と同様に、
  宣言によって(暗黙値に)初期化される訳ではない。
- Javaの場合は、プログラムのどこに変数宣言を置いてもよい。(但し、
  使う前に宣言しておかなければならない。)
- プログラムの途中で変数宣言を行う場合は、(変数の)名前の範囲に気
  を付けなければならない。例えば、

```
for (int i=0; i<n; i++) {
    [          ]
}
```

  と書くと、ループ制御変数 i はこのループの中だけで有効になる。

クラス, 局所変数, 定数変数に対する命名規則 : Sun コーディング規約

- クラス名, フィールド名, 変数名の付け方 (定数値領域を除く) :
  - ◇ 主として英小文字を用いる。
  - ◇ 2つ以上の「意味のある単語」から構成する場合は、2つ目以降の単語の頭文字を大文字にする。
  - ◇ クラス名の場合 は、先頭文字を大文字にし、名詞 (句) を表す単語列にする。
  - ◇ フィールド名, 変数名の場合 は、先頭文字を小文字にする。

- 定数値を保持するフィールド名, 変数名の付け方 :
  - ◇ 「意味のある単語」を構成する全ての英文字を大文字にする。
  - ◇ 2つ以上の「意味のある単語」から構成する場合は、(単語の区切を明らかにするために) 単語間にアンダースコア (_, 下線記号) を入れる。

例題18．1（円錐の体積）　2つの実数データ r, h を読み込み、
底面の半径が r、高さが h の円錐の体積
を出力するJavaプログラムを作成せよ。

（考え方）　例題1.2の計算処理をJavaで書いたらどうなるか、という例題である。

（プログラミング）ここでは、double型で計算処理するプログラムだけ

```
[motoki@x205a]$  cat -n VolumeOfCone.java   Enter
     1 /* 2つの実数データ r と h を読み込み、                    */
     2 /*        底面の半径が r、高さが h の円錐の体積   */
     3 /* を出力するJavaプログラム                              */
     4
     5 import java.util.Scanner;
     6
     7 public class VolumeOfCone {
```

```
 8      static final double PI = 3.1415926535897932;
                                                  /* 円周率 */
 9
10      public static void main(String[] args) {
11          Scanner inputScanner = new Scanner(System.in);
12
13          System.out.print("底面の半径： ");
14          double radius = inputScanner.nextDouble();
15          System.out.print("高さ： ");
16          double height = inputScanner.nextDouble();
17
18          System.out.printf("底面の半径が %f, " +
                              "高さが %f の円錐の体積%n" +
19                            "      = %f%n",
20                            radius, height,
                              PI*radius*radius*height/3.0);
21      }
```

```
    22 }
[motoki@x205a]$  javac VolumeOfCone.java  Enter
[motoki@x205a]$  java VolumeOfCone  Enter
底面の半径：  2.0  Enter
高さ：  5.0  Enter
底面の半径が 2.000000, 高さが 5.000000 の円錐の体積
     = 20.943951
[motoki@x205a]$
```

18–7 Scannerによる入力，printfメソッドによ

Scannerオブジェクトによるデータ入力：

Scannerクラスは、
書式付きファイルからのデータ入力を容易に行える様に
JDK1.6(JavaSE6.0,2006)で導入されたものである。

Scannerオブジェクトには、次の様なメソッドが他オブジェクトへのサービス窓口として用意されている。

メソッド名 … 説明
String nextLine() … 次の改行コードまでの文字列(改行コードは除く; 正確には文字列から作られたStringオブジェクトへの参照値)を返す。
String next() … 次のデータを構成する文字列(から作られたStringオブジェクトへの参照値)を返す。
int nextInt() … 次のデータを構成する文字列をint型に解読・変換し、その結果を返す。
long nextLong() … 次のデータを構成する文字列をlong型に解読・変換し、その結果を返す。
float nextFloat() … 次のデータを構成する文字列をfloat型に解読・変換し、その結果を返す。
double nextDouble() … 次のデータを構成する文字列をdouble型に解読・変換し、その結果を返す。

メソッド名 ··· 説明
boolean hasNext() ··· 次のデータを構成する文字列が来ているかどうかを調べ、その結果を返す。
boolean hasNextInt() ··· 次のデータを構成する文字列としてint型データとして解読できるものが来ているかどうかを調べ、その結果を返す。
boolean hasNextLong() ··· 次のデータを構成する文字列としてlong型データとして解読できるものが来ているかどうかを調べ、その結果を返す。
boolean hasNextFloat() ··· 次のデータを構成する文字列としてfloat型データとして解読できるものが来ているかどうかを調べ、その結果を返す。
boolean hasNextDouble() ··· 次のデータを構成する文字列としてdouble型データとして解読できるものが来ているかどうかを調べ、その結果を返す。
.....(他にも多数).....

System.out.printf()における書式指定の仕方：

Javaの場合は、変換指定は一般に次の形で行う。

%[引数インデックス][フラグ][最小フィールド幅][.精度]変換指定子

但し、[...] の部分は省略可。

この一般形の中で、

- "[引数インデックス]"の部分は 、変換指定に対応する引数を直接指定できる様にしたもので、1つの引数を複数回参照する場合に有用。
  次の2つの指定の仕方がある。
  - %引数番号$ ··· 何番目の引数に対応させるかを$文字の前に指定。
  - %< ··· 「1つ前の変換指定子と同じ引数」という意味

  引数番号の指定されないものについては、順に 1番目の引数，2番目の引数，... が対応付けられる。例えば、

```
System.out.printf("%3$d %d %2$d %<d %d %n", 1, 2, 3);
```

  というコードの出力は次の様になる。

```
3 1 2 2 2
```

- 変換指定子としては 次の様なものが用意されている。(日時の出力に

関しては、もっと多種類用意されている。詳しくは...）

変換指定子	説 明
d	byte, short, char, int型の引数に適用可能で、意味はC言語の場合とほぼ同じ。
o	byte, short, char, int型の引数に適用可能で、意味はC言語の場合とほぼ同じ。
x / X	byte, short, char, int型の引数に適用可能で、意味はC言語の場合とほぼ同じ。
c / C	文字変換で、byte, short, char, int型の引数に適用可能。指定されたデータがUnicode文字を表しているいるものと見て、その文字を出力する。
f	float, double型の引数に適用可能で、意味はC言語の場合とほぼ同じである。
e / E	float, double型の引数に適用可能で、意味はC言語の場合とほぼ同じ。
g / G	float, double型の引数に適用可能で、意味はC言語の場合とほぼ同じ。

変換指定子	説 明
a A	float, double型の引数に適用可能。 指定されたデータが浮動小数点数型の内部表現形式に従っているものと見て、それぞれ次の形式の指数部(底は2)付きの16進浮動小数点表記に変換して出力する。 [-] 0x 16進表記の小数 p [-] 底2の指数部,10進表記 [-] 0x 16進表記の小数 P [-] 底2の指数部,10進表記
b B	ブール値変換で、任意の引数に適用可。 引数が boolean 型ならその値に応じて "true" か"false" を出力。引数がnull なら "false", それ以外なら "true" を出力する。
h H	任意の引数に適用可で、ハッシュコード変換という。 引数がnullなら "null" を出力。それ以外なら、引数のhashCodeの結果をInteger.toHexString へ渡して得られる文字列を出力する。
s S	文字列変換で、任意の引数に適用可。 引数がnullなら "null" を出力。それ以外の場合、引数がFormattableインターフェースを実装していれば 引数.formatTo() という呼び出し, 実装してなければ 引数.toString() という呼び出しが行われ

変換指定子	説 明
tR	Dateオブジェクトの時刻部を例えば "09:55" という形で出力。
tT	Dateオブジェクトの時刻部を例えば "09:55:46" という形で出力。
tD	Dateオブジェクトの日付部を例えば "09/07/11" という形で出力。
tF	Dateオブジェクトの日付部を例えば "2011-09-07" という形で出力。
tc	Dateオブジェクトの内容を例えば "水 9 07 09:55:46 JST 2011" という形で出力。
n	C言語の場合と全く異なる。 Javaの場合は、プラットホーム固有の行区切り文字の出力を表し、必ずしも "\n"(line feed) という機能文字の出力と一致する訳ではない。
%	C言語の場合と同じ。

- "[最小フィールド幅]" の部分は 、C言語のprintfの場合と同じ意味
- "[精度]"の部分も 、C言語のprintfの場合とほぼ同じ意味
  - ◇ byte, short, int, long型に対しては指定不可。
  - ◇ float, double型の場合、f変換,e変換では小数点以下の桁数を指定し、g変換では有効桁数を指定する。
- "[フラグ]" の部分には 、次の様な指定が可能である。

フラグ	説 明
（なし）	右寄せ
-	左寄せ
+	o，x，X変換に対しては指定不可。また、空白フラグと一緒の使用は不可。 d，f，e，E，g，G，a，A変換の時、非負の値が + 符号付きで出力される。
空白（␣）	o，x，X変換に対しては指定不可。また、+フラグと一緒の使用は不可。 d，f，e，E，g，G，a，A変換の時、非負の値に対して先頭の符号部に空白が出力される。
#	d，g，G変換に対しては指定不可。 o変換の時、8進数の出力の前に 0 が付く。 x，X変換の時、16進数の出力の前に0xまたは0X が付く。 f，e，E，a，A変換の時、必ず小数点が付く。
0	出力フィールドを埋める文字として、空白ではなく 0（ゼロ）を用いる。

フラグ	説 明
,	o，x，X，e，E，a，A変換に対しては指定不可。 出力数値にグルーピング文字を含め、例えば 12345678 という整数に対しては 12,345,678 と出力する。
(	o，x，X，a，A変換に対しては指定不可。 引数が負の値の場合はマイナス符号を付ける代わりに括弧で囲んで出力する。

18–8 演算子

- C言語の場合とほとんど同じ。
- C言語で許されJavaで許されないものは次の通り。

ポインタに関する演算	`->`(メンバアクセス演算子), `*`(間接演算子), `&`(番地演算子)
構造体に関する演算子	`.`(メンバアクセス演算子)
その他	`sizeof`, `,`(コンマ演算子)

- C言語には無かったがJavaで導入されたものは次の通り。

  - .（ドット演算子） ··· どのオブジェクトのどのメソッド、
    といったことを表す時に用いる。
  - new（new演算子） ··· 新しくオブジェクトを生成する時に用いる。
  - instanceof（instanceof演算子）
    ··· 指定したオブジェクトが指定したクラスの
    インスタンスになっているかどうか、
    を判定する時に用いる。
  - >>>（右論理シフト） ··· 上位桁に必ず0を補う右シフト。

- C言語と少し違っているものは次の通り。

論理演算子 &&, ||, !	··· 論理型が設けられたことに伴う違いのみ 本質的な違いはない。
条件演算子 ? :	··· 論理型が設けられたことに伴う違いのみ 本質的な違いはない。
++(後置), --(後置)	··· Javaでは優先順位が最上位になった。 (前置のものについては2番目の優先順位 のまま。)
キャスト	··· Javaでは優先順位が1つ下がって 3番目になった。

優先順位高

演算子	結合性
メソッドの引数をくくる丸括弧 ( ) 配列添字をくくる四角括弧 [ ] ドット演算子 . ++(後置) --(後置)	左から右
+(単項) -(単項) ++(前置) --(前置) ! ビット反転~	右から左
new キャスト	右から左
* / %	左から右
+ -	左から右
左シフト<< 右算術シフト>> 右論理シフト>>>	左から右
< <= > >= instanceof	左から右
== !=	左から右
ビット積&	左から右
ビット排他的和^	左から右
ビット和|	左から右
&&	左から右
||	左から右
条件 ? 式 : 式	右から左
= += -= *= /= など	右から左

18–9 制御構造

C言語の場合と同様の、次の制御構造が使えます。(goto文は使えない。)

- if文
  if-else文
- while文
- for文　　(forループの中の初期化式，繰り返し式は
  コンマで区切った式のリストでもよい。)
- do-while文
- break文
- continue文
- switch文
- return文
- 条件演算子　(式1　? 式2　: 式3)

更に、次の制御構造も使える。

- 拡張for文：

  for (集合要素の型 集合要素を表す変数 : 集合を表すオブジェクト)

  「集合要素を表す変数」に対する操作；必要なら複合文に

(⟶ 18.12節)

例題18．2（Napier数eの計算）　ネピア数(Napier number)

$$e=\lim_{x\to\infty}(1+\frac{1}{x})^{x}=\sum_{i=0}^{\infty}\frac{1}{i!}=2.718281828\cdots\cdots$$

を小数点以下15桁まで計算するJavaプログラムを作成せよ。

（考え方）　$19! \approx 1.2 \times 10^{17}$ に注目して、

$$e \approx \sum_{i=0}^{18} \frac{1}{i!}$$

$$= (((\cdots((1 \cdot \frac{1}{18}+1)\frac{1}{17}+1)\cdots)\frac{1}{3}+1)\frac{1}{2}+1)\frac{1}{1}+1$$

という計算を行なえばよい。

（プログラミング）　この処理を行うクラスの名前を NapierNumber としてJavaプログラムを構成した。

```
[motoki@x205a]$  cat -n NapierNumber.java
     1 /* ネピア数 e=2.718281828... を                                */
     2 /*    e=((...(((1/18+1)/17+1)/16+1)...)/2+1)/1+1 */
     3 /* と計算するJavaプログラム                                    */
     4
     5  public class NapierNumber {
     6      public static void main(String args[]) {
     7          double e = 1.0;
     8          for (int i=18; i>0; i--)
     9              e = e / i + 1.0;
    10          System.out.println("e = " + e);
    11      }
    12  }
[motoki@x205a]$  javac NapierNumber.java
[motoki@x205a]$  java NapierNumber
e = 2.718281828459045
[motoki@x205a]$
```

口演習18．1（1000以下の完全数）　正整数 k が等式

k ＝ (kの約数の内、k以外のものの総和)

を満たす時、k は完全数であると言う。例えば、6の約数は 1,2,3,6 の4個であり 6=1+2+3 であるから、6 は完全数である。

1000以下の完全数を全て出力するJavaプログラムを作れ。

18–10 Mathクラス

- 17.3節で既に述べた様に、

  関連するデータ・機能(メソッド)をひとまとめにした
  (型枠)モジュール

  のことをクラスと呼んでいる。

- 自動的にimportされるjava.lang という標準パッケージ(i.e.ライブラリ)の中には、Mathというクラスが用意されており、そのクラスの中には各種数学関数を計算するためのメソッドが定義されています。

  ここで提供されているものは次の通りです。
  (C言語の数学ライブラリとほぼ同等のものが提供されている。)

機能	定数名	引数の型	定数値の型	説明
ネピア数	E	------	double	e=2.71828···
円周率	PI	------	double	π=3.14159···

機能	メソッド名 (引数の並び)	引数の型	関数値の型	説明
乱数	random()	なし	double	区間[0, 1)の間の疑似
切捨て	floor(a)	double	double	$\lfloor$ a $\rfloor$
切上げ	ceil(a)	double	double	$\lceil$ a $\rceil$
四捨五入	rint(a)	double	double	aの小数点以下を四捨五 (負数の時は 絶対値に対して四捨五
	round(a)	float	int	aの小数点以下を四捨五
	round(a)	double	long	aの小数点以下を四捨五
剰余	IEEEremainder (a, b)	double	double	a-b×rint(a/b)

機能	(引数の並び)	引数の型	関数値の型	説明
絶対値	abs(a)	int, long, float または double	引数の型と同じ	$\lvert a \rvert$
最大値	max(a, b)			a と b の最大値
最小値	min(a, b)			a と b の最小値
平方根	sqrt(a)	double	double	$\sqrt{a}$
べき乗	pow(a, b)	double	double	a^b
指数	exp(a)	double	double	e^a
自然対数	log(a)	double	double	$\log_e a$
正弦	sin(a)	double	double	sin a, 但し a
余弦	cos(a)	double	double	cos a, 但し a
正接	tan(a)	double	double	tan a, 但し a
逆正弦	asin(a)	double	double	$\sin^{-1} a \in [-\frac{\pi}{2},$
逆余弦	acos(a)	double	double	$\cos^{-1} a \in [0, \pi$
逆正接	atan(a)	double	double	$\tan^{-1} a \in [-\frac{\pi}{2},$
	atan2(a, b)	double	double	$\tan^{-1} \frac{a}{b} \in [-\frac{\pi}{2},$

- Mathクラスのメソッドを呼び出すには

  Math. メソッド名 (引数並び)

  という書き方をします。

- Mathクラスの定数を参照するには

  Math. 定数名

  という書き方をします。

- J2SE5.0(JDK1.5)以降では、次の様な宣言(staticインポート文という)をプログラムの先頭部(クラス定義等の前)に置くことにより ‘‘Math.’’ の部分を省略できる。

```
import static java.lang.Math.メソッド名 ;
import static java.lang.Math.定数名 ;
import static java.lang.Math.*;
```

例題18．3（円周率πの計算）　マチン(Machin)の公式

$$\pi = 16\tan^{-1}\frac{1}{5} - 4\tan^{-1}\frac{1}{239}$$

を用いて円周率を計算するJavaプログラムを作成せよ。

（考え方）　数学関数を呼び出すだけである。

（プログラミング）　比較のため`Math.PI`も表示...

```
[motoki@x205a]$  cat -n CircleRatio.java
   1 /* マチン(Machin)の公式を用いて円周率を計算するJava... */
   2
   3  public class CircleRatio {
   4    public static void main(String args[]) {
   5      double pi = 16.0 * Math.atan2(1.0,5.0)
                      - 4.0 * Math.atan2(1.0,239.0);
   6      System.out.println("Machin's formula ===>  pi = "
                               + pi);
   7      System.out.println("          Math.PI ===>  pi = "
                               + Math.PI);
   8    }
   9  }
[motoki@x205a]$  javac CircleRatio.java
[motoki@x205a]$  java CircleRatio
Machin's formula ===>  pi = 3.1415926535897936
         Math.PI ===>  pi = 3.141592653589793
[motoki@x205a]$
```

18–11 メソッド

- C言語の「関数」に相当するものは Javaではメソッドと呼ばれている。

メソッドに対する命名規則： 次の様な指針が一般的である。

- メソッド名の付け方：
  - ◇ 主として英小文字を用いる。
  - ◇ 2つ以上の「意味のある単語」から構成する場合は、
    2つ目以降の単語の頭文字を大文字にする。 Sunコーディング規約
  - ◇ メソッド名の場合 は、先頭文字を小文字にし、
    動詞(句)を表す単語列にする。
    Sunコーディング規約
  - ◇ フィールドの値を調べるメソッドの名前 は「‘‘get’’+フィールド名」にする。(e.g. getNum) 電通国際情報サービスコーディング規約
  - ◇ フィールドの値を設定するメソッドの名前 は「‘‘set’’+フィールド名」にする。(e.g. setValue) 電通国際情報サービスコーディング規約

例題18．4（素数の表）　1000以下の素数の表を作成するJavaプログラムを作成せよ。

（考え方）　例題3.10で作成したCプログラム（素数かどうかの判定）をJavaのメソッドとして作り替え、このメソッドを使って 2~ 1000 の数について順に素数かどうかの判定を行えば良い。

（プログラミング）

```
[motoki@x205a]$  cat -n PrimeNumbers.java
     1 /* 1000以下の素数の表を作成するJavaプログラム */
     2
     3 public class PrimeNumbers {
     4     private static final int MAXPRIME = 1000;
     5     private static final int WIDTH    = 10;
     6
```

```
 7      public static void main(String args[]) {
 8          System.out.println("Table of prime numbers");
 9          int count = 0;
10          for (int k=2; k<=MAXPRIME; k++) {
11              if (prime(k)) {
12                  System.out.printf("%5d", k);
13                  count++;
14                  if (count >= WIDTH) {
15                      System.out.println();
16                      count = 0;
17                  }
18              }
19          }
20          if (count > 0)
21              System.out.println();
22      }
```

```
23
24      // 引数kが素数かどうかを判定する
25      public static boolean prime(int k) {
26          int limit = (int) Math.sqrt( (double) k );
27          for (int i=2; i<=limit; i++) {
28              if (k%i == 0)
29                  return false;
30          }
31          return true;
32      }
33 }
[motoki@x205a]$  javac PrimeNumbers.java
[motoki@x205a]$  java PrimeNumbers
Table of prime numbers
    2    3    5    7   11   13   17   19   23   29
   31   37   41   43   47   53   59   61   67   71
   73   79   83   89   97  101  103  107  109  113
```

（中略）

```
  877   881   883   887   907   911   919   929   937   941
  947   953   967   971   977   983   991   997
[motoki@x205a]$
```

18–12 配列

- Javaでは、配列はオブジェクト(i.e.関連するデータ・機能をひとまとめにしたモジュール)として扱われる。

- Javaでは、配列そのものに固定的な名前が付けられている訳ではなく、配列を参照する変数を通して各配列要素にアクセスする。

  [C言語で、malloc関数で配列領域を確保し、そこへのポインタを変数に記憶する、といった感じ。]

  ⇨ 配列を参照する変数は(動的に)色々な配列を参照できる。

例18．5(配列を参照する変数)　C言語では配列そのものに名前が付けられ、それがコンパイル時に固定的に使われるが、Javaにおいては配列そのものに固定的な名前が付けられる訳ではない。このことは次のJavaプログラムによって理解されよう。

```
[motoki@x205a]$  cat -n ArrayObject.java
     1 /* Javaにおける配列の扱い方を説明するためのプログラム例 */
     2
     3 public class ArrayObject {
     4     public static void main(String args[]) {
     5         int[] a123 = {1, 2, 3};
     6         int[] b45  = {4, 5};
     7
     8         int[] tmp = b45;       // swap
     9         b45       = a123;
    10         a123      = tmp;
    11
    12         for (int i=0; i<b45.length-1; i++)
    13             System.out.print(b45[i] + ", ");
    14         System.out.println(b45[b45.length-1]);
    15     }
    16 }
[motoki@x205a]$  javac ArrayObject.java
[motoki@x205a]$  java ArrayObject
1, 2, 3
[motoki@x205a]$
```

拡張for文（for-each構文）: 配列を始め、要素の何らかの集合を表すオブジェクトがあった時、配列や集合の各々の要素に対して操作を加えたいことがある。この様な場合、次の形式で繰り返し処理を表すことができる。

```
for ( 集合要素の型 集合要素を表す変数 : 集合を表すオブジェクト )
    「集合要素を表す変数」に対する操作；必要なら複合文に
```

例題18．6（平均と分散） 50個の実数データ $x_0, x_1, x_2, \cdots, x_{49}$ を読み込み、それらの平均 μ と分散 V を定義式

$$\mu = \frac{1}{50}(x_0 + x_1 + x_2 + \cdots + x_{49})$$

$$V = \frac{1}{50}\sum_{i=0}^{49}(x_i - \mu)^2$$

に従って求めて出力するJavaプログラムを作成せよ。

（考え方） 例題1.4の計算処理をJavaで書いたらどうなるか、という

例題である。

（プログラミング）　50個の実数データ x_0, x_1, x_2, ... , x_{49} を保持するために x という名前のdouble型配列を用意し、

$$x_0 + x_1 + x_2 + \cdots + x_{49}, \qquad \sum_{i=0}^{49} (x_i - \mu)^2$$

の累算を一旦 sum という名前のdouble型変数上に行うことにしてプログラムを構成した。
この計算を行うJavaプログラムと、これをコンパイル/実行している様子を次に示す。

```
[motoki@x205a]$  cat -n AverageVariance.java
     1 /* 50個の実数データ x0, x1, x2, ... , x49 の平均 mu と
     2 /* 分散 V を定義式
     3 /*    mu = (x0+x1+x2+ ...  + x49)/50
     4 /*     V = (x0-mu)^2 + (x1-mu)^2 + ... + (x49-mu)^2 / 50
     5 /* に従って求め、それらの値を出力するJavaプログラム
     6
     7 import java.util.Scanner;
```

```
 8
 9 public class AverageVariance {
10     public static void main(String args[]) {
11         double[] x = new double[50];
12         Scanner inputScanner = new Scanner(System.in);
13
14         for (int i=0; i<x.length; i++)
15             x[i] = inputScanner.nextDouble();
16
17         double sum = 0.0;
18         for (double element : x)
19             sum += element;
20         double average = sum/(double)x.length;
21
22         sum = 0.0;
23         for (double element : x)
24             sum += (element-average)*(element-average);
```

```
25          double variance = sum/(double)x.length;
26
27          System.out.printf("%nInput data:%n");
28          for (int i=0; i<x.length; i++) {
29              System.out.printf("%14.6e", x[i]);
30              if (i%5 == 4)
31                  System.out.println();
32          }
33          System.out.printf("%nAverage  = %14.6g%n" +
34                            "Variance = %14.6g%n",
35                            average, variance);
36      }
37 }
[motoki@x205a]$  javac AverageVariance.java
[motoki@x205a]$  java AverageVariance < ../../Programs-C/
                                  fundamentals-ave-var.data
```

```
Input data:
  1.000000e+00  1.000100e+00  1.000200e+00  1.000300e+00  1.00
  1.000500e+00  1.000600e+00  1.000700e+00  1.000800e+00  1.00
  1.001000e+00  1.001100e+00  1.001200e+00  1.001300e+00  1.00
  1.001500e+00  1.001600e+00  1.001700e+00  1.001800e+00  1.00
  1.002000e+00  1.002100e+00  1.002200e+00  1.002300e+00  1.00
  1.002500e+00  1.002600e+00  1.002700e+00  1.002800e+00  1.00
  1.003000e+00  1.003100e+00  1.003200e+00  1.003300e+00  1.00
  1.003500e+00  1.003600e+00  1.003700e+00  1.003800e+00  1.00
  1.004000e+00  1.004100e+00  1.004200e+00  1.004300e+00  1.00
  1.004500e+00  1.004600e+00  1.004700e+00  1.004800e+00  1.00

Average  =         1.00245
Variance =    2.08250e-06
[motoki@x205a]$


---
```

例題18．7（エラトステネスのふるい）　「エラトステネスのふるい」アルゴリズムを用いて1000以下の素数の表を作成するJavaプログラムを作成せよ。

（考え方）　エラトステネスの篩アルゴリズムでは、
最初に 2 から 素数かどうかを調べたい最大の整数 までの整数の集合を素数の候補の集合 S として構成する。そして、素数の候補 k=2，3，4，...，素数かどうかを調べたい最大の整数 の順に次の(1),(2)を繰り返す。

(1) k ∈ S かどうかを調べ、もし k ∈ S なら k は素数、
　　もし k ∉ S なら k は素数でないと判断する。

(2) もし k ∈ S で k が素数と判断できるなら、

　　(2.1) 素数の候補の集合 S に篩をかけ、S から k 以外の k の倍数を取り除く。すなわち、

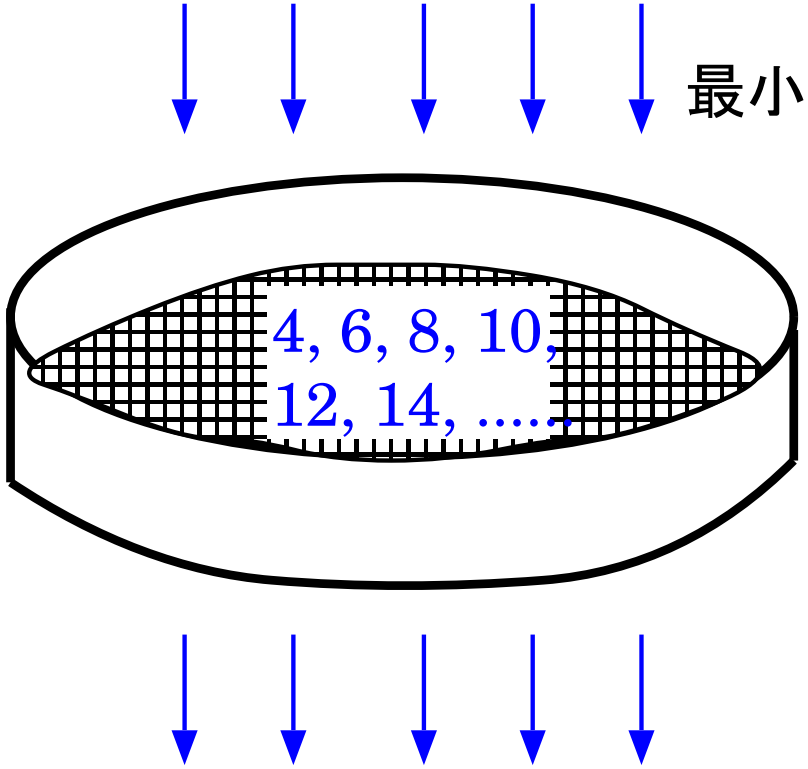
$$S \longleftarrow S - \{2k,\ 3k,\ 4k,\ 5k,\ \ldots\}$$

(2.2) k を素数として出力する。

素数の候補

2, 3, 4, 5, 6, 7, 8, 9, 10,
11, 12, 13, 14, 15, ...

最小要素2は素数



最小要素以外を篩いにかける

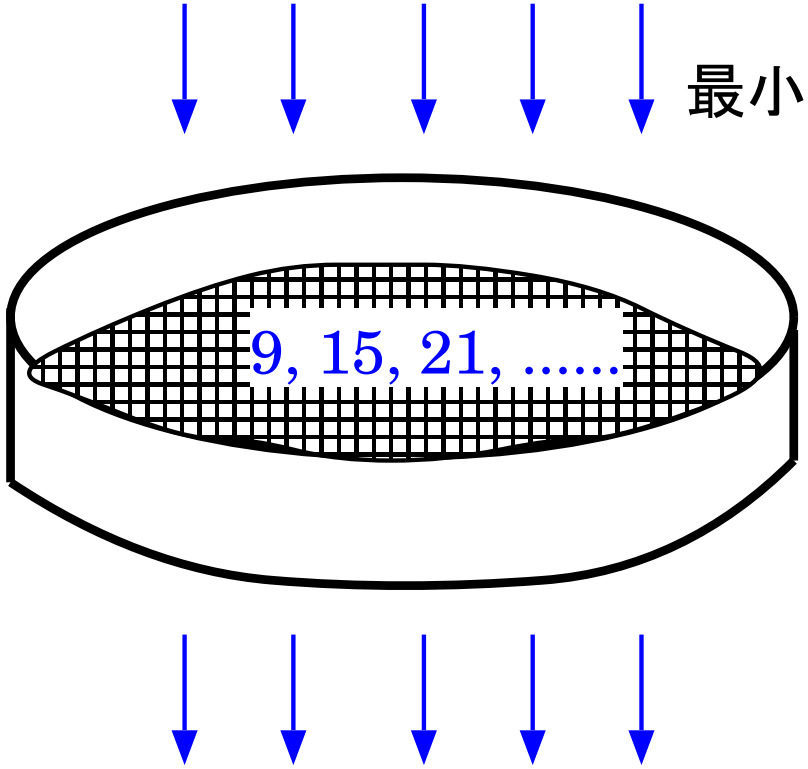
最小要素2の倍数を通さない篩

3, 5, 7, 9, 11, 13, 15,
17, 19, 21, ...

素数の候補

3, 5, 7, 9, 11, 13, 15,
17, 19, 21, ...

最小要素3は素数



5, 7, 11, 13, 17,
19, ...

（プログラミング）　素数の候補の集合 S の中には 2 以上，
素数かどうかを調べたい最大の整数 以下の整数しか入らないので、

S を boolean 型の配列で表す

ことが出来る。

実際、配列 prime[0]~ prime[1000] を用意して、

(∀ k)[k∈S ⟺ prime[k] = true]

という風に S を表せば良い。

```
[motoki@x205a]$  cat -n EratosthenesSieve.java
     1 /*  「エラトステネスのふるい」アルゴリズムを用いて */
     2 /* 1000以下の素数の表を作成するJavaプログラム      */
     3
     4 public class EratosthenesSieve {
     5     private static final int MAXPRIME = 1000;
     6     private static final int WIDTH    = 10;
     7
     8     public static void main(String args[]) {
```

```
 9          boolean[] prime = new boolean[MAXPRIME+1];
10          for (int k=2; k<prime.length; k++)
11              prime[k] = true;
12
13          System.out.println("Table of prime numbers:");
14          int count = 0;
15          for (int k=2; k<prime.length; k++) {
16              if (prime[k]) {
17                  System.out.printf("%5d", k);
18                  count++;
19                  if (count >= WIDTH) {
20                      System.out.println();
21                      count = 0;
22                  }
23                  for (int i=k+k; i<prime.length; i+=k)
24                      prime[i] = false;
25              }
26          }
```

```
    27          if (count > 0)
    28              System.out.println();
    29      }
    30 }
[motoki@x205a]$  javac EratosthenesSieve.java
[motoki@x205a]$  java EratosthenesSieve
Table of prime numbers:
    2    3    5    7   11   13   17   19   23   29
   31   37   41   43   47   53   59   61   67   71
   73   79   83   89   97  101  103  107  109  113

                  (中略)

  877  881  883  887  907  911  919  929  937  941
  947  953  967  971  977  983  991  997
[motoki@x205a]$
```

配列についてのまとめ

- 配列を参照する変数の宣言は次の様に行う。

  データ型 [] ···[] 変数名 ;
  クラス名 [] ···[] 変数名 ;
  データ型 変数名 []···[] ;
  クラス名 変数名 []···[] ;

- 配列オブジェクトを生成して、既に宣言された(配列)変数がその配列を参照するように設定するには、次のようにする。

  変数名 = new データ型 [要素数] ··· [要素数] ;
  変数名 = new クラス名 [要素数] ··· [要素数] ;

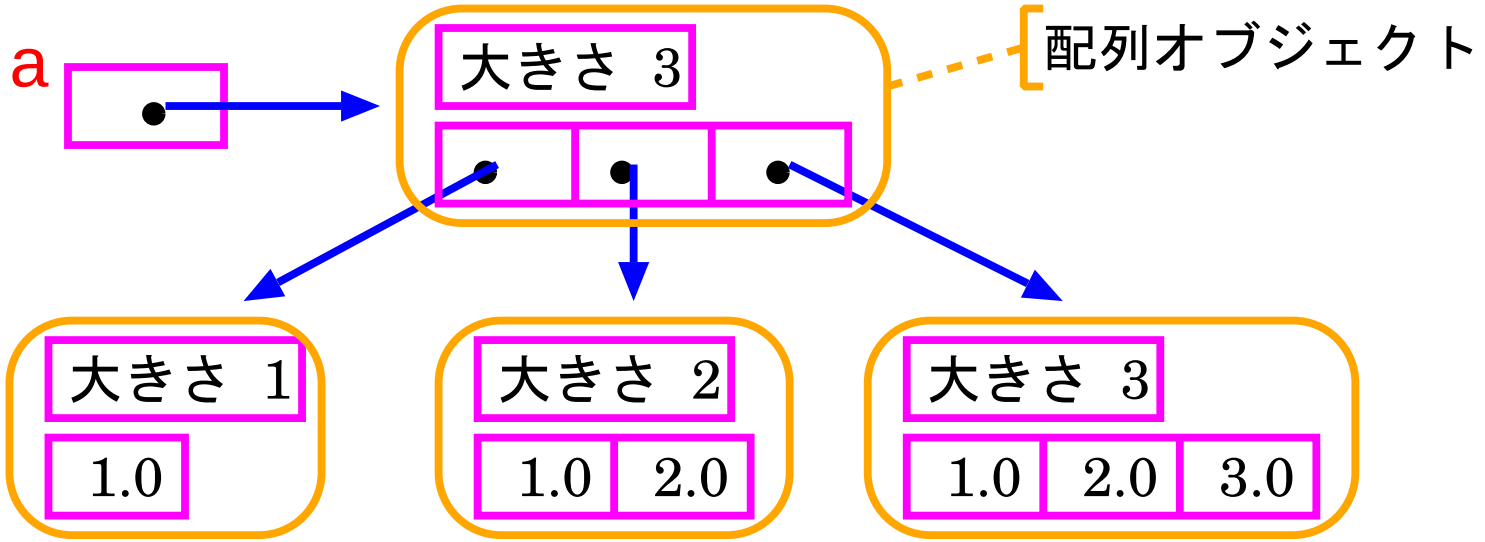
  もちろん、この作業を、配列を参照する変数の宣言と同時に行うこともできる。 例えば、

```
int[] a = new int[10];
```

- 配列の配列の場合は、小配列の大きさはバラバラでも良い。

- 配列オブジェクトを生成する時に初期設定も行う場合は、C言語の場合と同様の書き方が許されます。例えば、

```
double[][] a = { {1.0},
                 {1.0, 2.0},
                 {1.0, 2.0, 3.0} };
```



- 配列オブジェクトの大きさ情報はその配列オブジェクト自身が持っている。（固定されている。）

  配列の大きさ情報にアクセスするには次の様に書く。

  配列の参照`.length`