JAVA 入門

横田 壽

目次

まえがき

生命情報，生体認証，バイオコンピューティングなど，21 世紀の学問と呼ばれる新しい分野が成長してきています．これらの分野は，コンピュータ科学の発展に伴って生まれてきたものです．コンピュータ科学は，大雑把に分けると，アルゴリズム，データ構造，データベース，アーキテクチャ，ネットワーク，グラフィックス，ソフトウェア，セキュリティなどからなる学問です．そして，これらのコンピュータ科学の発展を支えたのは，数学，物理学，電子・電気工学です．

コンピュータ科学の発展に寄与したもう 1 つの要因に，プログラミング言語とコンパイラがあります．このテキストを手にしている人の多くは，すでになんらかのプログラミング言語を学んだことがあると思います．しかし，問題は，何人がすでに学んだプログラミング言語で，自分のやりたいことをプログラムに直し，実行することができますか，ということです．

もう 1 つの問題は，これから学ぶプログラミング言語は，オブジェクト指向でなければならないということです．そこで，このテキストでは，オブジェクト指向言語としての構造をもつ JAVA 言語について学べるようになっています．さらに，JAVA 言語を単に言語として学ぶだけでは，身に付かないという経験から，アルゴリズムとデータ構造を 15 回の講義で学べるようになっています．

最後に本書がコンピュータ科学への入門書としての役割を果たし，各専門分野での勉学の架け橋となることを願いま

著者

第 1 章

JAVA プログラミングを始めよう

Java の基礎 Java プログラムは，**ソースコードとオブジェクトコード**の 2 つの形態があります．ソースコードはプログラムのテキスト版で，テキストエディタを用いて作成したものです．オブジェクトコードはプログラムの実行形態です．オブジェクトコードはコンピュータ上で実行することができます．C/C++ などのオブジェクトコードは，特定の CPU 専用で，異なるプラットフォームでは実行できません．しかし，Java はプラットフォーム非依存です．

Java はプラットフォーム非依存を実現するために，コンパイル時にバイトコードと呼ばれる特定の CPU に依存しない命令を含んだオブジェクトファイルを生成します．このバイトコードは **Java Virtual Machine(JVM)** によって解釈できるように設定されています．

Java のプラットフォーム非依存の鍵は，同じバイトコードをあらゆるプラットフォーム上の JVM で実行できる点にあります．

この章では，JAVA 言語を用いた標準出力とよばれる画面 (コンソール) への文字や数字の出力方法について学びます．このようなアプリケーションをコンソールアプリケーションといいます．Java ではこの他に，GUI(グラフィック・ユーザ・インターフェイス) を用いた**アプリケーションとアプレット (applet)** があります．アプリケーションは，JVM によって直接実行することができます．しかし，アプレットを実行するには web ブラウザが必要です．

出力メソッド (Output Method)

Java には System.out.println() という画面に表示するための**出力メソッド (output methods)** が用意されています． このメソッドは引数として渡された文字列を出力し，その後改行文字を出力します．他にも System.out.print() メソッドもありますが，こちらは改行文字を出力しません．セミコロンはステートメントの終わりを示します．

```
System.out.println(”Hello”);
```

によって，2 重引用符” ”内の文字はそのまま標準出力に渡され表示されます．出力する文字列を途中で改行 (new line) するには ¥n と 2 重引用符内で書きます．また，段落 (tab) を取るには, 2 重引用符内で ¥t と書きます．

例題 1.1 画面への出力

Hello　world

　　初めての Java

と画面に表示しなさい．ただし，初めての Java の前には段落をとるとします．

解答 Java のプログラムでは必ずクラス名を決めなければなりません．ここではクラス名を Hello とします．Java では applet 以外のプログラムでは，**main() メソッド**とよばれるものも必要となります．メソッドとは，あるまとまった処理を行なう一連のプログラムで，{ から } までが 1 つのメソッドとなります．また， Java で画面へ出力するには，System.out.println() メソッドを用います．そこで，次のようなプログラムを書きます．

```
==========================================================
class Hello
{
        public static void main(String[] args)
        {
            System.out.println(”Hello world ¥n ¥t 初めての Java ¥n”);
        }
}
==========================================================
```

実行結果

このプログラムをエディタ (vi，emacs，秀丸，MKEditor，メモ帳等の文書を書く機能を持ったもの) を使って書き，Hello.Java という名前を付けて保存します．Java ではファイル名はクラス名と同じでなければなりません．

次に，java のコンパイラを用いてコンパイルします．JDK をインストールした場合，JDK をインストールしたディレクトリに移動し，次のコマンドをコマンドプロンプトで入力します．

javac Hello.java

エラーが発生しなかったら，次に

java Hello

とコマンドプロンプトで入力します．

コンパイラの仕事の一つはプログラムのテキストが正確かを解析することです．コンパイラはプログラムの意味は理解できないが，プログラムの構文については間違いを見つけることができ

ます．よくある二つのプログラムエラーは次のようなものです．

1. 構文エラー
2. 型エラー

構文エラーとは，例えば次のようなものです．
public static void main(String[] args; //) がない
{
　　System.out.println(”Hello Java”): // 違法文字’:’
}
型エラーとは，用いる変数の型に応じて型を宣言しなければならないのに，間違った型を宣言してしまったりすることをいいます．

エラーメッセージは一般に式番号とコンパイラがたどりついた簡単なエラーの発生理由を含んでいます．エラーは発生順に訂正する習慣を身に着けるべきです．エラーの修正が済んだらまたコンパイルします．このステップはエディットーコンパイルーデバッグとよばれます．

コンパイラの仕事のもう一つの役割は，コード生成とよばれるコンピュータに分かる命令テキストにプログラムを変換することです．Java コンパイラは，実行可能コードを生成するのではなく，**バイトコード (byte code)** を含んだオブジェクトファイルを作成します．バイトコードとは，特定の CPU に依存しない命令のことです．バイトコードは **Java Virtual Machine(JVM)** によって解釈できるように設計されています．このことにより，Java はプラットフォーム非依存を実現させています．

コンパイルがうまくいくとバイトコードが作られるので，Java インタープリタを呼び出してプログラムを実行することができます．プログラムを実行するには

java Hello

と入力します．

プログラム Hello.java の解説 (Explanation of Hello.java)

- 1 行目の class Hello は，Hello クラスの宣言です．Java では全てがクラスを用いて行われます．
- 2 行目の public static void main(String[] args) はメインメソッドで，public は外からのアクセスが可能であることを示します．
- static はこのメソッドは静的であり，このメソッドはこのクラスに属しており，このクラスから作られるオブジェクトには属さないことを表しています．
- void はこのメソッドの戻り値は何もないことを意味しています．
- main はメソッド名で，そのあとに続く String[] args の String は文字列型とよばれ，文字列を操作するときに用います．String[]　args で変数 args は文字列の引数を何個かとる配列であることを表しています．

- { から } までが main メソッドの中身です.
- System.out.println は標準出力（ディスプレイ）へ出力させるためのメソッドです．println 内の ¥n　は行を変えろということを意味し，¥t はタブを取れということを意味します．これらを**エスケープ文字 (escape sequence)** といい，2 重引用符内からの脱出を図るときなどに用います.
- Java のプログラムは，物理的な行に制約を受けないフリーフォーマット (自由書式) ですが，プログラムが読みやすいように空白を入れたり，字下げ (インデント) などを行います．ただし，全角の空白は空白と判断されずエラーの元になるので注意が必要です.
- Java のプログラムは小文字で記述します．Java のプログラムでは大文字と小文字は区別され，異なる文字とみなされます．ただし，記号定数だけは一般の変数と区別するために，大文字で書く慣わしとなっています.
- Java のプログラムでは，セミコロン (;) のついた 1 つの命令をステートメントといいます．ステートメントの最後には，必ずセミコロンを記述します．また，セミコロンで区切ることにより，1 行に複数の文を書くことができます.

クラスとオブジェクト

クラスとオブジェクトは，すべての Java プログラムの構築ブロックを形成しています．したがって，これらについて基本的なことを理解しておく必要があります.

オブジェクトとは，状態と動作の両方を定義する記憶領域のことです．記憶領域はメモリであることもディスクであることもあります．状態は一連の変数とその中の値によって表現されます．動作は一連の**メソッド**とそれによって実装されるロジックによって表現されます．したがって，オブジェクトとは，データとそれを操作するコードの組合せであると言えます.

クラスとは，オブジェクトを作成するための雛形 (テンプレート) のことです.

練習問題 1.1 コンソールへの出力

こんにちは
　　誰々さん
と表示するプログラムを作成しなさい．ただし，誰々さんには自分の名前を入れます.

エスケープ文字 エスケープ文字には，次のようなものがあります.

エスケープ文字	機能
¥b	カーソルを 1 文字分戻す
¥f	次のページの先頭にカーソルを移動する (改頁)
¥n	次の行の先頭にカーソルを移動する (改行)
¥r	同じ行の先頭にカーソルを移動する (復帰改行)
¥t	次の水平タブ位置にカーソルを移動する (水平タブ)
¥v	次の垂直タブ位置にカーソルを移動する (垂直タブ)
¥¥	¥ 記号を示す
¥’	引用符を示す
¥”	2 重引用符を示す
¥0	ナル値を示す
¥DDD	8 進数 DDD で表される文字を示す
¥uxxxx	Unicode 文字（xxxx は 4 桁の 1 6 進数定数)

例題 1.2　出力メソッドへの数値の挿入

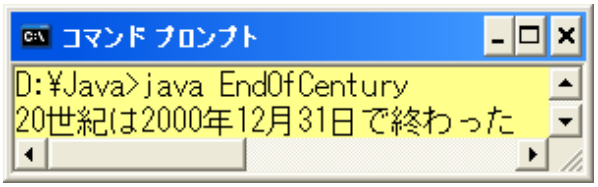
次の文章を出力しなさい.

２０世紀は２０００年１２月３１日で終わった.

解答

```
==========================================================
class EndOfCentury
{
        public static void main(String[] args)
        {
            System.out.println(20 + ”世紀は” + 2000 + ”年” + 12 + ”月” +
            31 + ”日で終わった”);
        }
}
==========================================================
```

実行結果



当然，System.out.println(”20 世紀は 200 年 12 月 31 日で終わった.”); と書くこともできます．ここでは，Java の文字列の扱いがいかに柔軟であるかを示すために，このような書き方をし

てみました．数字と文字を + 演算子で結ぶと，数字を文字列に直し連結してくれます．

練習問題 1.2 出力メソッドへの数値の挿入
自分の生年月日を出力しなさい．

文字とリテラル (Characters and Literals)

文字とは文章を書く文字に 1 0 個の数字と句読点を含めたものをいいます．文字は整数としてコンピュータに格納されます．格納方法で最も有名なのが **ASCII コード (ASCII code)** です．ASCII は American Standart Code for Information Interchange の略で，英数字を ISO コードにパリティビットをつけた 8 ビットで表します．Java では Unicode を用いています．Unicode は ASCII コードをそっくり取り込んでいるので，半角文字に関しては，ASCII 文字感覚で用いることができます．

例えば大文字の A は ASCII コードでは，10 進数で 65 として格納されます．ちなみに，65 は 2 進数で表すと

$$(65)_{10} = 2^6 + 1 = (1000001)_2$$

16 進数で表すと

$$(1000001)_2 = 0100|0001 = (41)_{16} = 0\text{x}41$$

8 進数では

$$(1000001)_2 = 001|000|001 = (101)_8 = 0101$$

となります．

Unicode では，大文字の A は 10 進数で 0065 として格納されます．また，漢字を用いるときもそれが Unicode で表現されていることを意識する必要はありません．

データ型の具体的な値，つまり定数のことを**リテラル (literal)** といいます．例えば，文字型 (char) のリテラルの例として'A'，String 型のリテラルの例として上の例題の"20 世紀"，整数型のリテラルとして 2000 などがあります．

改行文字 '¥n' は余白文字 (whitespace character) の 1 つで，表示されない文字の 1 つです．'¥n' で 1 つの文字を表しています．これと同じ方法で作られている文字には，'¥t' (タブ) などがあります．

変数とその宣言 (Variables and Their Declarations)

変数はコンピュータのメモリへの格納場所を表す識別子です．その場所に格納された情報を変数の値といいます．変数は使用する前に必ず宣言されなければなりません．変数の値を取り出すには，代入が用いられその構文は

変数 = 式;

となります．最初に式が求められ，その結果が変数に代入されます．ここで，等号"="はプログラミング言語では代入演算子です．

例えば，メモリに格納された整数型の変数 m の値 5 を表示するには，

```
int m = 5;
System.out.println(m);
```

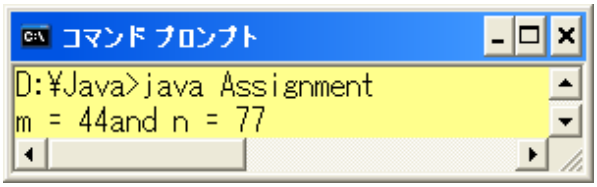
と書くことができます．

例題 1.3 数値の代入
整数 44 を変数 m に代入し，式 m + 33 の結果を n に代入するプログラムを作成しなさい．

解答 整数 44 を変数 m に代入するには，まず，整数 44 を格納するための場所と名前が必要なので，int m; と宣言します．同様に，n には m と整数 33 の合計が格納されるので，int n; の宣言も必要です．出力は 2 重引用句の中は記された通りに出力されることを利用し，System.out.println(”m=” + m); とすれば，何を出力しているのか分かりやすくなります．これより，次のようなプログラムとなります．

```
==========================================================
class Assignment
{
        public static void main(String[] args)
        {
            int m,n;
            m = 44;
            System.out.println(”m = ”+ m);
            n = m + 33;
            System.out.println(” and n = ” + n);
        }
}
==========================================================
```

実行結果



練習問題 1.3 数値の代入
整数 44 を変数 m に代入し，式 m/12 の結果を予想し，自分の予想が正しいかプログラムを書いて調べなさい

変数 m と n は次のように考えることができます．

これはまるで郵便受けのようです．変数名 m が住所で，44 は郵便受けの中身，そして型 int が郵便の種類（速達，書留など）です．

この例題で，変数 m,n は int m,n; と宣言されていますが，同じ型の変数はこのようにまとめて宣言することができます．

Java プログラムでは，全ての変数は使う前に宣言されていなければなりません．その記述法は

[型修飾子] データ型 変数名

となります．

オブジェクト指向言語では，メソッド内で用いられる変数を**ローカル変数 (local variable)** といい，クラスのメンバ変数を**フィールド (field)** といいます．

変数の名前

変数名はその変数が何を表しているのかが分かりやすい名前を付けます．そして，その名前の付け方には，次のような規則があります．

- 名前は文字 (A ∼ Z, a ∼ z) で始める．
- 名前に用いられる数字と文字の長さに制限はありません．
- アンダースコア (_) と$を用いることができる．

これとは，別に Java スタイルというものがあり，

height_of_box よりも heightOfBox と表す**キャメルケース**を用いる．

変数の初期化 (Initializing Variables)

変数の初期化とは，初めて変数の宣言をしたときに変数に値を代入することです．多くの場合，変数を宣言するときに同時に初期化を行っておくほうが賢明です．変数の初期化は，

int x = 1024;

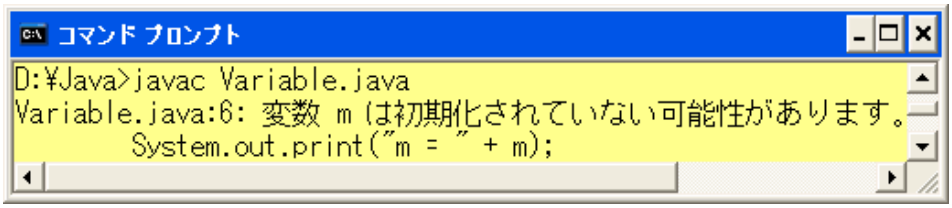
のように書きます．

例題 1.4 初期化していない変数

変数を初期化せずに用いた場合，次のプログラムは何を表示するか調べなさい．

```
==================================================
class Variable
{
        public static void main(String[] args)
        {
            int m;
            System.out.println("m = " + m);
        }
}
==================================================
```

実行結果



Java では，初期化されていない変数はコンパイルエラーとなります．このことにより，C 言語や C++ 言語で問題であった隠れたエラーの原因が一つ解消されます．C 言語と異なり Java 言語では，変数は使う直前で定義します．そのことにより，プログラムが読みやすくなります．

プログラムトークン (Program Token)

コンピュータプログラムは**トークン (token)** と呼ばれる要素の並びです．トークンは int などのキーワード，main などの識別子，{ などの句読点，'+' などの演算子を含んでいます．このプログラムをコンパイルするとき，コンパイラはソースコードを走査し，構文解析します．もし，構文エラーを見つけると，コンパイルをやめエラーメッセージを出します．例えば，文の終わりにセミコロンを付けるのを忘れたりすると，セミコロンが欠けているとのメッセージを出します．しかし，括弧を閉じるのを忘れたり，ダブルコーテーションの 2 つ目を忘れたりした場合は，コンパイラはプログラムのどこかがおかしいとのエラーメッセージを出すだけで，何がおかしいのかは自分で見つけなくてはなりません．

例題 1.5 プログラムトークン
次のプログラムには何個のトークンがあるか調べなさい．

```
=========================================================
public static void main(String[] args)
{ // "n = 44"を表示します．これはコメントです．
/* これもコメントで，複数行にまたがるときに用います．*/
/** これもコメントで，これは javadoc というソフトウェアをもちいてドキュメントを
生成するときに用います**/
        int n=44;
        System.out.println("n = " + n);
}
=========================================================
```

解答 このソースコードには 28 個のトークンがあります．"public", "static", "void", "(", "String", "[", "]", "args", ")", "{", "int", "n", "=", "44", ";", "System", ".","out",".", "println", "(", "n" , "=", "+", "n", ")",";", "}"

コンパイラはコメントは無視するので，トークンの数には含まれません．

定数 (Constant)

Java では変数は値を持った箱と理解すると分かりやすいでしょう．箱の名前が変数名で，箱の中身が値と考えます．変数名にはなるべく中身が分かりやすい名前を付けます．例えばテストの平均を入れる箱には testAverage という名前をつけます．この箱の中身が変えられない物を定数といいます．定数はその型の前にキーワード **final** をつけます．final が付くとこの変数はこれが最後の形となり，その後値を変えることができなくなります．つまり，定数ということです．定数は，宣言されるときに初期化もされなければなりません．

例題 1.6　定数の指定
次のプログラムは定数の定義のしかたを表しています．

```
=====================================================
class UsefulConstant
{
        public final int MAXINT = 2147483647;
        public final float KM_PER_MI = 1.60934f;
        public final int SCREEN_WIDTH = 640;
}
=====================================================
```

定数は大文字であらわすのが決まりになっています．

整数と実数 (Integers and Real Numbers)

Java では数値を実数と整数で区別します．もっと詳しくいうと，1/2 と 1/2.0 ではまったく違う値になります．1/2 は 0 になり，1/2.0 は 0.5 になります．これは，/という記号が商を表すということで理解できます．

Java の基本データ型の詳しい紹介は 2 章で行ないますが，プログラミングは”習うより慣れろ”なので，幾つか紹介しておきます．

例題 1.7　基本データ型（=の両側にはスペースをいれるようにする）
基本データ型を調べなさい．

```
char  c  = 'A';          /* 文字型*/
boolean b = true;        /* 論理型*/
byte b = 1 ;             /* バイト型 (−128 ∼ 127)*/
short k = 32456;         /* 整数型（−32, 768 ∼ 32, 767）*/
int i = 2442343434;      /* 整数型 (−2^31 ∼ 2^31 − 1 の 1 0 進整数)*/
long l = 1234L;          /* 長い整数 (最後尾に L または l を付ける)*/
float f = 124.56f;       /* 実数型（小数点または 1 0 の累乗で表し，単精度）*/
double d = 3.1415E3;     /*実数型（float と同形式）d=3141.5 を表す*/
```

char は character（文字）の略，int は integer（整数）,float は float(浮動小数点) のことです．Java では long 型の値には末尾に L または l をつける必要があります．float も末尾に f をつけ

ます.

浮動小数点出力 (Floating Point Output)

次のような小数点を含んだデータ X,Y,Z があります.

	X[i]	Y[i]	Z[i]
1	12.24	13.456	234.2345
2	23.345,	32.12,	26.1235
3	432.2,	67.3,	45.2331

これを

```
for(int i=1;i<=3; i++){
        System.out.println(X[i] + " " + Y[i] + " " + Z[i]);
}
```

と書くと,

12.24	13.345	234.234
23.345	32.12	26.1235
432.2	67.3	45.2331

と表示されます.

表記 (Representation)

Java では 10 進数（decimal)，8 進数 (octagonal)，16 進数 (hexadecimal)，指数 (exponent) の表記が可能です．8 進数の表記には 0 を数字の前に付け，16 進数では 0x を付ける．例えば，10 進数 40 を 16 進数に直すと

$$40 = 2 \times 16 + 8$$

となるので 0x28 となります．Java では,

```
System.out.println(0x28);
```

と書くと，40 と表示されます．では，10 進数を 40 を 16 進数で表すには，どうすればよいでしょうか．Java のライブラリには，Integer ラッパークラスとよばれるものがあり,

```
int m = 40;
Integer.toString(m,16);
```

と書くことにより，16 進数に変換できます．この他にも,

```
String m = "40";
Integer.parseInt(m,16);
```

と書くことにより，16 進数で 40 になる数に変換できます．

実行結果

練習問題 1.4 表記

10 進数 40 を 16 進数と 8 進数で出力するプログラムを作成しなさい．

クラスとメソッドの基礎

クラスとオブジェクトは Java のプログラムの構築ブロックであるといいました．そこで，ここでは，クラスとオブジェクトがどのように作成され用いられるのか見てみます．クラスを理解するために，例として図形を考えましょう．図形には色々ありますが，例えば三角形を思い浮かべると，三角形には底辺と高さがあり，これによって面積が求まります．この三角形の属性である底辺と高さ，また，面積を求める方法などを 1 つのまとまりとしたものを三角形クラスといいます．このとき，底辺や高さを**フィールド** (フィールド変数) といい，面積を求めるための動作を**メソッド**といいます．

この三角形クラスの宣言は，識別子 class のあとにクラス名がきて，中括弧 { から中括弧 } までが本体となります．

クラス宣言 (Class Declarations)

クラス宣言を用いて上の事柄を宣言すると次のようになります．

```
=====================================================
class Triangle
{
        private double base, height; //(底辺，高さ)
        public void assign(double b, double h) // 代入メソッド
        {
            base = b;
            height = h;
        }
        public double area()
        {
            return base*height/2;
        }
        public void display()
        {
```

```
            System.out.println("底辺" + base + "高さ" + height + "の三角形の面積は" + area());
        }
}
==========================================================
```

ここで，class はクラス宣言を示すキーワードで，次の Triangle はこのクラスの名前です．{と} に囲まれた中は，クラスのフィールドとメソッドと呼ばれ，メンバどうしの名前が重複しなければ，どのような名前でも使えます．また，クラスのメンバはディフォルトでは非公開 (private) メンバになります．したがって，クラスの外から参照する必要のあるコンストラクタやメソッドは公開 (public) にする必要があります． 公開にするには public を付けます．逆に，フィールドは原則非公開 (private) にする必要があります．なぜならば，フィールドが公開になっていると，クラスの外部からフィールド値の変更が可能となり，コントロールができなくなってしまいます．このように，フィールドを非公開にして，外部からアクセルできなくすることを**カプセル化 (encapsulation)** といいます．これは，オブジェクト指向プログラミングの 3 つの原則の 1 つです．

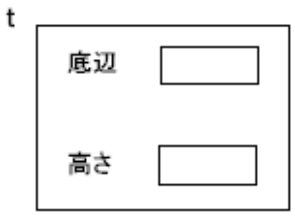
クラスのオブジェクトを作る

このクラスを実際に使うには，変数を宣言して，その実体を用意する必要があります．この実体を**オブジェクト**といいます．Triangle t = new Triangle() で，t は Triangle クラスのオブジェクトとして宣言されます．よって，t 自身がフィールド変数，base と height を持っています．また，オブジェクト t はメソッド assign(), area(), display() をよぶことができます．ここで，注意して欲しいのは， メンバ関数 display() は，その持ち主の名前をつけて，t.display() というように用いられます．

実際，メソッドはこのように用いるしかありません．

オブジェクト t は普通の変数のように宣言されます．この型は Triangle です．これは**ユーザ定義型** (抽象データ型) として考えることができます．

Java では，int , float, double などの型に Triangle 型を付け加えることができます．これはオブジェクト t を次のように考えることで理解できます．



また，オブジェクト t を宣言することを，オブジェクトの**インスタンス化 (instanciation)** といい，オブジェクト t を Triangle クラスのインスタンスといいます．インスタンスとは，そのクラスの**特定事例**だと考えれば分かりやすいでしょう．

クラスを用いたプログラム

例題 1.8 三角形クラス
底辺 18，高さ 24.5 の三角形の面積を求めるプログラムを Triangle クラスを用いて作成しなさい．

解答

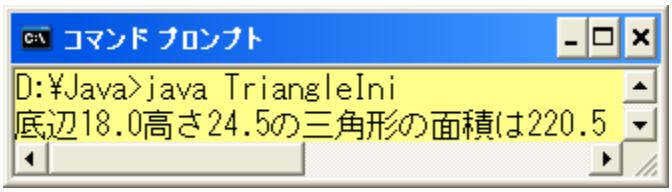
1. Triangle クラスの宣言を行います．フィールドは base と height とします．
2. メソッドの定義を行います．メソッドの定義は
   戻り値のデータ型　メソッド名 (仮引数の型　仮引数)
   {
   }
   で行います．したがって，
   public void assign(double b, double h)　{
   　　base = b;
   　　height = h;
   }
   と定義すると，Triangle クラスのフィールド変数 base に assign メソッドで引いてきた値 b が代入され，height に h の値が代入されます．これより，base と height を Triangle クラスの他のメソッドで用いることができます．
3. 最後に main メソッドを用いるためのクラス TriangleIni クラスを作成し，main メソッド内で Triangle クラス型のオブジェクトを作り，三角形の面積を求めます．

```
==================================================
class Triangle
{
        private double base, height;
        public void assign(double b, double h)
        {
            base = b;
            height = h;
        }
        public double area()
        {
            return base*height/2;
        }
        public void display()
        {
            System.out.println("底辺" + base + "高さ" + height + "の三角形の面積は" + area());
```

```
        }
}
public class TriangleIni
{
        public static void main(String[] args)
        {
            Triangle t = new Triangle();
            t.assign(18,24.5);
            t.display();
        }
}
```

==========

実行結果



この例題から分かるように，Java のプログラムを学ぶには，メソッドについて学ばなければなりません．次の章からはメソッドで用いる順接，選択，反復について学んだあと，メソッドの用い方について学んでいきます．

確認問題 1.1

1. Java の 3 通りコメントの方法を示せ.

2. 次の C スタイルのコメントはどこがいけないのか説明しなさい.

```
System.out.println("Hello, /* 変更 */ World.¥n");
```

3. 宣言は何をするものか説明しなさい.

4. Java で最も短いプログラムは何か.

5. 次の宣言はどこがいけないか

```
int first = 22, last = 99, new = 44, old = 66;
```

6. 次の文の誤りを見つけ，正しなさい.

```
int double=44;
```

7. ASCII コードが 100 である文字を見つけるためにはどんなプログラムを書けばよいか

8. 次のプログラム中の予約語，識別子，演算子，リテラル，句読点シンボル，コメントを捜し出しなさい.

```
class Exercise
{
    public static void main(String[] args)
    {
      int n;
      n *= 3; // n = n*3
      System.out.println("n = " + n);
    }
}
```

9. 次の 2 つの文の違いを説明しなさい.

```
char c = 'A';
char c = 65;
```

10. a の ASCII コードを見つけるコードを書きなさい.

演習問題 1.1

1. 次のプログラムの誤りを見つけなさい.

```
class UndeclaredInt
{
    public static void main(String[] args)
    {
      // "n=22"を表示：
      n = 22;
      System.out.println("n =" + n);
```

```
    }
  }
```

2. 次のような実行結果を得られるように，プログラムの中の□を埋めて，プログラムを完成しなさい．

Main は最初に呼び出されるメソッド．
　　System.out.println は表示するためのメソッド．

```
=====================================================
class FillBlank
{
  public static void main(String[] args)
  {
    System.out.println("Main は最初に呼び出されるメソッド．
    □ System.out.println は表示するためのメソッド．") ;
  }
} =====================================================
```

3. 以下の項目で，実数型で表す必要のあるものはどれか．

ア　体重の平均を計算します．

イ　人数の合計をカウントします．

ウ　実行回数をカウントアップします．

4. 以下の空欄を埋めよ．

10 進数	16 進数	8 進数
1 8	ア	イ
ウ	0x25	エ
オ	カ	012
8	キ	ク

5. 以下に Java の構文で表した数字があります．同じ値のものどうしのグループに分けよ．

.25e2, 0x25, 25, 012, 37, 8, e1, 8.e0

第 2 章

データの基本型

キーワード (予約語)(Keywords)

「キーワード」は Java コンパイラに対して，データ型や命令文の意味を持ち，ユーザはこれを識別子としては使えません．Java のキーワードには以下のものがあります．覚える必要はありません．

abstract	boolean	break	byte	case
cast	catch	char	class	compl
const	continue	default	do	double
else	extends	false	final	finally
float	for	future	generic	goto
if	implements	import	inner	instanceof
int	interface	long	native	new
null	operator	outer	package	private
protected	public	rest	return	short
static	super	switch	sychronized	this
throw	throws	transient	true	try
var	void	volatile	while	

識別子 (Identifier)

「識別子」はユーザが「クラス」「変数（メンバ変数)」「メソッド（メンバ関数)」につける名前です．名前には「文字」「数字」「_」「＄」が使えますが，キーワードと同じ名前は使えません．また，名前の先頭文字には「数字」，「_」は使えません．識別子に付ける名前は慣習的に次のようになっています．

識別子の型	慣習	命名例
クラス名	各語の先頭は大文字， 単語をつないで表記します．	TextEditor ErrorDialog
メンバ関数名	先頭語を小文字， 後の語の先頭を大文字で表記します．	addFisrt() hasNext()
メンバ変数名	メンバ関数名と同じ．	dayofWeek summerVacation
定数	すべて大文字で表記し，	PI

基本データ型 (Basic Data Types)

Java では変数を入れる箱の型を決める必要があります．箱の型を決める前に箱の名前，つまり変数名を決めなければなりません．変数名は中身にあった名前を付けるのが良いとされています．例えば，テストの回数を表す変数ならば，count や testCount や testTimes などが良い名前です．

ここでは，変数 testTimes がテストの回数を表すとしましょう．この場合，テストの回数は 1,2,3…という整数なので，整数をしまっておく箱は整数型であると宣言します．例えば，

```
int testTimes;
```

と書きます．また，test_Average がテストの平均だとすると，小数になる可能性があります．そこで，小数をしまっておく test_Average は，実数型であると宣言します．例えば，

```
float test_Average;
```

と書きます．しかし，テストの回数が 2 の 32 乗を超えるようであれば，int ではまずいことになります．そのときは long を用います．このように，箱に入る中身によってそれに見合った箱を用意できるようになっています．Java などのオブジェクト指向プログラミングでは，すべてが型で構成されているので，マスターしておきましょう．

型	基本型			
	論理型	文字型	整数型	浮動少数点型
	boolean	char 'A'	byte short int long	float double

このあと，例題を交えながら基本データ型について細かく見ていきます．

論理型 (Boolean Type)

論理型は false または true の 2 値しかとることができないときに用います．

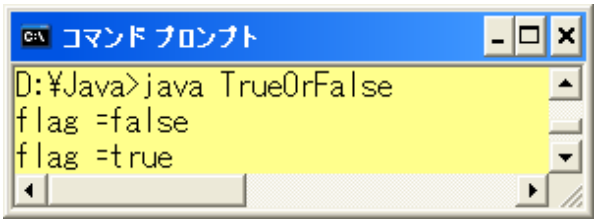
例題 2.1 論理型

次のプログラムを実行するとどんな結果が得られるか調べなさい．

```
class TrueOrFalse
{
        public static void main(String[] args)
        {
            bool flag = false;
            System.out.println("flag = " + flag );
            flag = true;
            System.out.println("flag =" + flag);
        }
}
```

解答 Java では false, true は予約語で，false は 0，true は 1 がプリントされます．

実行結果



文字型 (Character Types)

文字型は文字'A' や桁数'8' などをあらわすときに用います．例えば，A という文字を文字型 c に代入するときには，

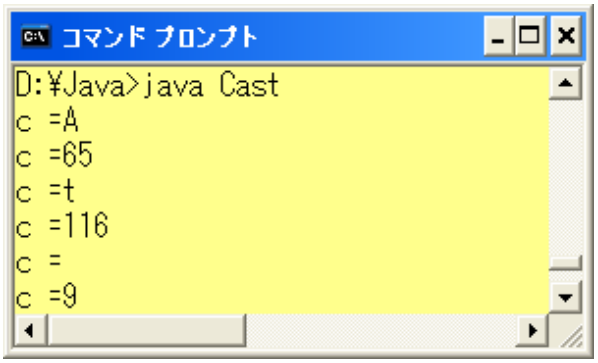
```
char c = 'A';
```

と書きます．文字 A を変数ではなく文字型として使用するときは，アポストロフィー (') が必要です．アポストロフィーなしでは変数と見なされ，エラーとなります．そこで，変数と区別するためにすでに学んだリテラルを用います．ここで，(int)c; とすると， (int)c は文字 A の ASCII コードである 65 を表します．これは**キャスト (cast)** とよばれ，Java では変換後の型をカッコで囲み変換したい変数の前に書きます．

例題 2.2 文字型
次のプログラムの結果はどうなるか考えなさい．

```
class Cast
{
        public static void main(String[] args)
        {
            char c = 'A';
            System.out.println("c =" + c );
            System.out.println("c =" + (int)c );
            c = 't';
            System.out.println("c = "+ c);
            System.out.println("c = "+ (int)c);
            c = '¥t';
            System.out.println("c = "+ c);
            System.out.println("c = "+ (int)c);
        }
}
```

解答 文字'A' を表示しようとすると，整数 65 ではなく文字"A"が表示されます．
実行結果



練習問題 2.1 文字型

文字 Y の ASCII コードを出力するプログラムを作成せよ．

整数型 (Integer Types)

Java には 4 個の整数型が用意されています．

例題 2.3 整数型の演算

Java では次の 5 つのオペレータで算術の計算を行います．+, - ,* , /, %．そこで，次のプログラムの結果はどうなるか考えなさい．

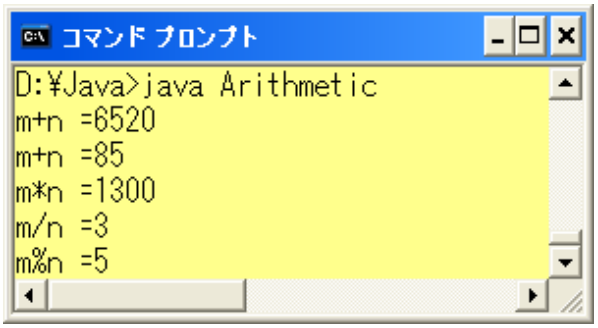
```
==========================================================
class Arithmetic
{
```

```
    public static void main(String[] args)
    {
        int m = 65;
        int n = 20;
        System.out.println("m+n = " + m + n);
        System.out.println("m+n = " + (m + n));
        System.out.println("m-n = " + (m - n));
        System.out.println("m*n = " + (m * n));
        System.out.println("m/n = " + (m / n));
        System.out.println("m%n = " + (m % n));
    }
}
```

==========

解答 System.out.println("m+n =" + m + n); では'+' が左から順に処理されるので，最初の +m で整数 65 が 6 と 5 の並びとして処理され，その後，20 が 2 と 0 の並びとして結合されるので，6520 となります．残りの問題は，カッコ内を先に処理するため，演算子として処理されます．他に注意しなければならないのは m/n は商を表し，m%n はあまりを表すことでしょう．

実行結果



練習問題 2.2 整数型の演算

整数 1234 の末尾を消して，123 を取り出すプログラムを作成しなさい．

増加と減少演算子 (Increment and Decrement Operators)

++ と −− 演算子は演算を行います．数の前におくか後ろにおくかで意味が異なることに注意しましょう．

例題 2.4 増加演算子

前増加と後増加の違いを確認しなさい．

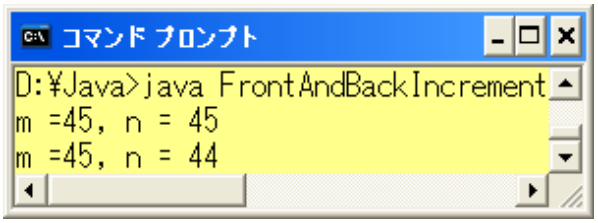
==========

```
class FrontAndBackIncrement
{
        public static void main(String[] args)
       {
            int m,n;
            m = 44;
            n = ++m;
            System.out.println("m = " + m + ", n = " + n );
            m = 44;
            n = m++;
            System.out.println("m = " + m + ", n = " + n);
        }
}
```

==========

解答 n = ++m は先に m の値を 1 増やし，それを n に代入するので前増加といいます．また n = m++ は m の値を先に n に代入し，その後に m の値を 1 増やすので後増加といいます．++m も m++ も，ともに m = m+1 のことであるが，m = m+1 とはほとんど書かないので，++ と −− の使い方に慣れておきましょう．

実行結果



練習問題 2.3 増加演算子

int m=44 のとき，m/++m の値はいくつになるか考え，プログラムを組んで確認しなさい．

浮動小数点型 (Floating-Point Types)

実数を表すのに用いられる型です．例えば 10 進数 13.75 が与えられたとします．これをコンピュータはどう扱うか見てみよう．まず，10 進数 13.75 は 2 進数に変換されます．

$$13.75 = (8+4+1) + (1/2 + 1/4) = 1101.11$$

次に，少数点をいちばん左まで移動します．つまり 4 ビットの移動を行います．ので，

$$13.75 = +0.110111_2 \times 2^4$$

書き直すと

$$13.75 = 0110111010000_2$$

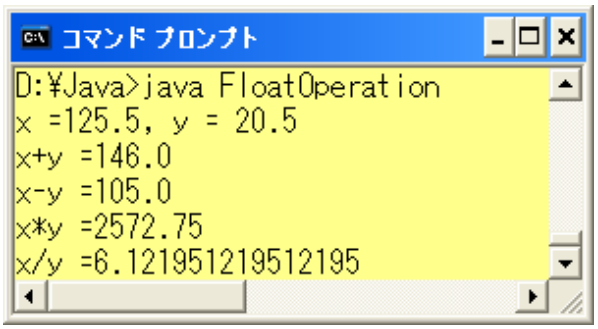
このとき，最初の 0 は符号部が正であることを表し，次の 6 桁は少数部（仮数部）を表し，次の 6 桁は指数部を表します．指数の最初の数字は符号を表します．一般に 32 ビットの浮動少数点型では，最初の 1 ビットが符号部，次の 23 ビットが少数部，残りの 8 ビットが指数部を表しています．ちなみに 64 ビットの double 型では，最初の 1 ビットが符号部で，少数部には 52 ビットが割り当てられ，11 ビットが指数部に割り当てられます．

浮動小数点型の算術計算は，整数型と異なり +,-,*,/の 4 種類で，普通の少数の計算と同じです．

例題 2.5 浮動小数点型の演算
次のプログラムの結果はどうなるか考えなさい．

```
=====================================================
class FloatOperation
{
        public static void main(String[] args)
        {
            double x = 125.5;
            double y = 20.5;
            System.out.println("x = " + x + "and y = " + y );
            System.out.println("x+y =" + (x+y));
            System.out.println("x-y =" + (x-y));
            System.out.println("x*y =" + (x*y));
            System.out.println("x/y =" + (x/y));
        }
}
=====================================================
```

実行結果



浮動小数点算術はすべて double で行われるので，double 型の代わりに float を使うときは，保管場所とアクセスタイムを気にするくらい多くの実数を使うときだけです．

ここからは，基本形ではないので Java がはじめてのプログラミング言語の人は，さっと読んで

おくだけでよいでしょう．6 章から 9 章で詳しく取り扱います．

配列型 (Array Type)

配列はオブジェクトです．つまり，配列は **new** 演算子を用いて新たに作成しないと使うことができません．配列の宣言は

型名 [] 配列名

で行なわれます．配列の宣言は データ型　配列名 [] と書くこともできますが，推奨できません．

例えば，int[] args と書くと args という配列名は配列型であることを意味します．この配列に 5 個の整数型の要素を持たせるには，

int[] args = new int[5];

と書きます．**new** 演算子により新たに配列を作成するには，要素の型と個数を new の後に書きます．これにより，args[0], args[1], args[2], args[3], args[4] の 5 つの要素が作成されます．配列の要素の番号は 0 から始まるので，要素数が 5 ということは 0 から 4 までとなります．

配列を初期化する方法は 2 通りあります．1 つは，次の例題で示します．

例題 2.6 配列への格納
整数 {2,3,5,7,11,13,17} を prime という配列名を持った配列に格納しなさい．

解答

int[ ] prime = {2,3,5,7,11,13,17};

と書けばよい．

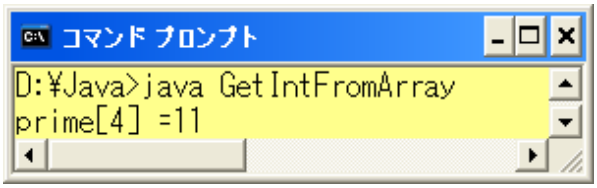
例題 2.7 配列の要素
整数 {2,3,5,7,11,13,17} の 5 番目の要素 11 を k という変数に代入し表示しなさい．

解答

```
==========================================================
￥class GetIntFromArray
{
        public static void main (String[] args)
       {
            int[] prime= {2,3,5,7,11,13};
            int k = prime[4];
            System.out.println(”prime[4] =” + k);
        }
}
==========================================================
```

と書けばよいでしょう．配列は 0 から始まるので prime[4] が 5 番目の 11 となります．

実行結果



練習問題 2.4 配列の要素

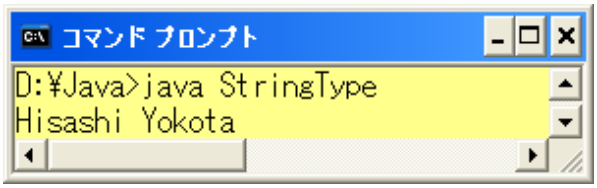
{0,1,1,2,3,5,8,13,21,34} を要素とする配列 a を定義し，第 3 番目と 4 番目の要素の和を求めるプログラムをしなさい．

文字列型 (String Tyep)

配列は同じ型の要素をまとめて扱うときに便利です．しかし，文字を配列として扱うよりも，文字を扱う型を用意したほうが便利さが増します．そこで，Java には **String 型**とよばれる文字列を扱う型があります．String 型は次のようにして用いることができます．

```
==========================================================
class StringType
{
        public static void main(String[] args)
        {
            String ss = ”Hisahi”; // 宣言と初期化
            ss = ss + ” Yokota”;
//ss = ”Hisashi Yokota”                System.out.println(ss);
        }
}
==========================================================
```

実行結果



誤差 (Error)

コンピュータのレジスタが有限桁数で演算をおこなうため，レジスタに入りきらない数値があると無視されたりして，演算結果と真の値に違いが出る．このような違いを**誤差 (Error)** といいます．

オーバーフロー (Overflow)

ほとんどのマシンでは，long int 型は 4 バイトつまり，32 ビットの領域が与えれています．しかし，これを超えるような計算をやらせると，間違った結果を表示します．これを**オーバーフ**

ロー (**overflow**) といいます．

アンダーフロー (Underflow)

データ型で許されている最小値より小さい絶対値を持つ値が代入され，零になることを**アンダーフロー (underflow)** といいます．

まるめ誤差 (Round-Off Error)

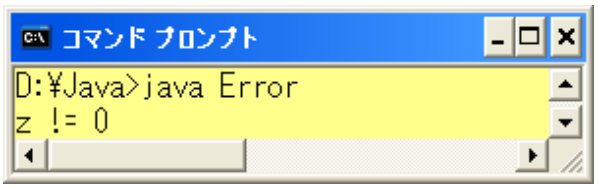
例えば，有理数 1/3 をコンピュータは 0.333333 としてメモリにしまう．このとき生まれる誤差をラウンドオフ エラーまたは**まるめ誤差 (round-off error)** といいます．

例題 2.8 まるめ誤差
次のプログラムを実行するとどうなるか調べよ

```
==========================================================
class Error
{
        public static void main(String[] args)
        {
            double x = 1000/3.0;
            double y = x - 333.0;
            double z = 3*y -1.0;
            if (z == 0) System.out.println("z = 0");
        else System.out.println("z != 0");
        }
}
==========================================================
```

解答 このプログラムを実行すると，z != 0 が表示されます．紙の上での計算では z=0 のはずがコンピュータでは z!=0 になります．つまり，まるめ誤差のため，z は 0 とはなりません．つまり，プログラムで値が零なのかテストするには，z == 0 ではなく，$|z| < 10\text{E-}20$ などでテストしましょう．したがって，1 点でのテストは控えるようにしましょう．条件文については次の章で説明します．

実行結果



練習問題 2.5 まるめ誤差
次のプログラムを実行し，まるめ誤差がどこに潜んでいるのか見つけなさい．

```
class RoundError
{
public static void main(String[] args)
   {
     double x = 1/9.0;
     double y = 1/11.0;
     double z = x - y;
     double w = 99*z;
     System.out,println("w = " + w);
     if(w == 2) System.out.println("w = 2");
     else System.out.println("w != 2");
   }
}
```

けた落ち (Cancelling in Digits)

ほぼ等しい数値どうしの引き算，絶対値がほぼ等しく符号が異なる 2 数の加算などを行った場合，有効桁数が急激に減少することがあります．このような現象を**けた落ち (cancelling in digits)** といいます．

$$345.678 - 345.666 = 0.012$$

の計算では，6 桁の有効桁数が引き算することにより，わずか 2 桁に減少しています．

情報落ち

大きな値と小さな値の加減算を行った場合に，小さい値のけたを大きな値のけたにそろえることによって，仮数部に入りきらない小さい値の情報の一部が落ちてしまうことがあります．このような現象を**情報落ち**といいます．

$$123456 + 12.3456 = 123468$$

となるが，実際の値は 123468.3456 です．したがって，この計算では 12.3456 の 0.3456 の部分が計算結果に何の影響も及ぼさありません．

スコープ (scope) 宣言を行います．ことにより識別子は使えるようになります．このとき，識別子はプログラムテキストの特定部分でしか使えません．例えば，関数内で宣言された識別子の場合は，スコープは，宣言の位置から宣言が行われたブロックの末尾までです．ここで，**ブロック (block)** とは，{ } で区切られた部分です．

確認問題 2.1

1. count が 100 を超えたら，”多すぎます” と表示する文を 1 行で書きなさい．

2. 次の式の値を求めなさい．ただし，m の値は 25，n の値は 7 とします．

1. m - 8 - n
2. m = n = 3
3. m%n
4. m%n++
5. m&++n
6. ++m - n−−

3. n から 1 を引くというコードを 4 つの方法で表しなさい．

4. どこが間違っているか．

a. if (x < y) min = x
　else min = y;

演習問題 2.1

1. 次のプログラムの実行結果がどうなるか考えなさい．

```
class CharType
{
        public static void main(String[] args)
        {
            char c = 'A';
            int i;
            int j = 0x41;
            System.out.println("¥nc=" + c);
            i = (int)c;
            System.out.println("¥nc=" + i);
            System.out.println("¥nj=" + j);
            i= 66;
            System.out.println("¥ni=" + i);
            c=(char)i;
            System.out.println("¥nc=" + c );
        }
}
```

2. 10 個の大文字の母音と小文字の母音を表示するプログラムを例題 2.2 をまねて作成しなさい．

第 3 章

選択 (selection)

代入演算子 (assignment operator)

int a=13 ;

と書くと，a という変数名がついた箱の中に 13 という整数値が記録されます．図で表すと

のような感じです．

このとき用いた”=”は代入演算子とよばれ，代入記号の右側の値を左側の変数に代入することを意味します．数学の等号ではないことに注意が必要です．等号の左側に来ることができる値を**左側値 (lvalue)** といいます．例えば，

int n;

n = 44;

は可能ですが，

44 = n;

はエラーです．

複合代入演算子 (Composite Assignment Operator)

演算子	例	意味
+=	a += b	a+b を a に代入
-=	a -= b	a-b を a に代入
*=	a *= b	a*b を a に代入
/=	a /= b	a/b を a に代入
%=	a %= b	a%b を a に代入

練習問題 3.1 複合代入演算子に慣れる

a = 33, b = 44 のとき，a*b の結果を a に代入し表示するプログラムを複合演算子を用いて作成しなさい．

型変換 (Type Conversions)

```
int n = 22;
float x = 3.141592;
x += n;
System.out.println(x - 2);
```

において，x += n; により，整数 n=22 は自動的に n=22.0 に変換されます．また，x - 2 において，2 は 2.0 に変換されます．このように，小さな幅の型から大きな幅の型への変換は自動的に行なわれますが，大きな型から小さな型へ変換は自動的ではありません．

一般に，T がある型で，v が別の型の値であるとき，

```
(T)v
```

により，v は型 T に変換されます．このことを**キャスト (cast)** といいます．キャストは，変数もしくは定数の前につけることで型の変換を行います．

```
(型) 変数　(型) 定数
```

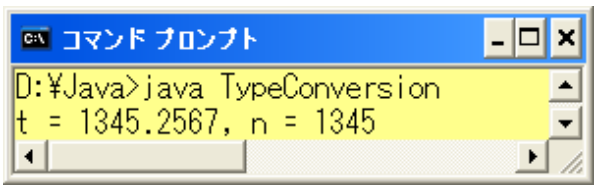
例えば，n が浮動小数点型のとき，(int)n と書くと，n は小数点以下を全て無視して整数型になります．

例題 3.1 キャスト

次のプログラムは double 値を int 値に**キャスト**します．

```
==========================================================
class TypeConversion
{
public static void main(String[] args)
{
        double t = 1345.2567;
        int n = (int)t;
        System.out.println("t = " + t + ", n = " + n );
        }
}
==========================================================
```

実行結果



小さい型から大きい型への変換は**プロモーション (promotion)** といい，そのための演算子は何も必要としありません．

入力ストリーム (Input Streams)

キーボードからの入力（標準入力）を取り込むには，System.in という入力ストリームを用いる方法があります．System.in は InputStream オブジェクトを指しています．その唯一のデータ入力メソッドである read() は，バイトコード入力という基本的な機能だけを持っています．つまり，

```
int idt = System.in.read();
```

によって，1 文字読み込み，その文字コードを idt に代入します．また，複数文字読み込むには，

```
byte[] bdt = new byte[80];
int nn = System.in.read(bdt,0,80);
String name = new String(bdt,0,nn-2);
```

を用います．最初の 2 行で，キーボードで Enter キーが押されるまで，文字列を byte 型配列 bdt の先頭から格納し，実際に読み込んだ文字数を nn に代入します．残りの 1 行は，byte 型配列 bdt に格納された文字列を読み込んだ文字より 2 文字少なく文字列 str に代入します．2 文字少なくする理由は，read() メソッドで読み込んだ後，改行文字 (Windows 環境で，0D 0A のペア文字) が格納されるためです．また，System.in の標準入力を用いるには，java.io.*ライブラリをインポートしなくてはなりません．ライブラリのインポートは，

```
import java.io.*;
```

で行います．

例外処理 プログラムを作成する場合，例外処理が必要な場合があります．例外処理とは「普通ではない処理」のことで，

- 特別な状態になった
- エラーが発生した

の二つの意味を含みます．ここで，「特別な状態」とは，キーボード入力時の Enter キー入力の発生などのことで，エラーではありませんが，何かの対処を必要とするものです

そこで，この例題では，入力時のキーボード入力を検査するため，try catch の構文で「特別な状態」を捕まえる処理が必要となります．

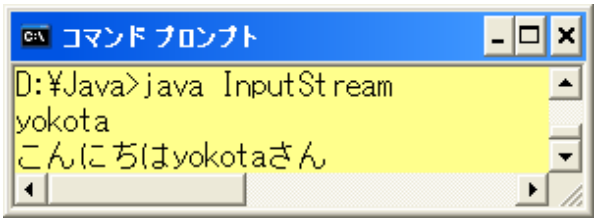
例題 3.2　キーボードからの入力
コマンドラインから自分の名前を打ち込んだら次の行に”こんにちは誰々さん”と表示するプログラムを考えなさい．

解答 キーボードから入力した名前を保存するには，配列，文字列と Java には色々用意されています．キーボードから入力した名前は System.in.read(bdt,0,80); で byte 型配列 bdt に格納され

ます．String name = new String(bdt,0,nn) で byte 型配列 bdt に格納された文字を nn 個文字列型変数 name に代入します．では，プログラムを作成しましょう．

```
import java.io.*;
class InputStream
{
        public static void main(String[] args)
        {
            byte[] bdt = new byte[80];
            try
            {
                System.out.println("名前はなんですか ?" );
                int nn = System.in.read(bdt,0,80);
                String name = new String(bdt,0,nn-2);
                System.out.println("こんにちは" + name + "さん");
            }
            catch(IOException ex)
            {
                System.out.println(ex);
            }
        }
}
```

実行結果



この例題では，名前をコマンドラインから入力しました．それに対して，次の例題では表示させるものをコマンドライン引数として使います．Java の main メソッドの args[0] は第 1 引数となります．

例題 3.3　コマンドライン引数

自分の名前を引数として打ち込んだら次の行に"こんにちは誰々さん"と表示するプログラムを考えなさい．

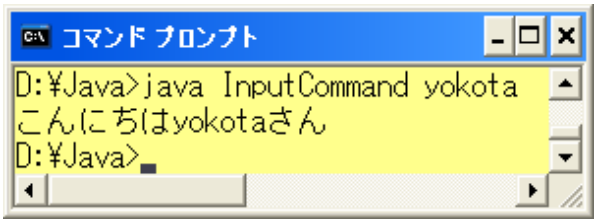
解答 main(String[] args) において第 1 引数は，argv[0] に代入されます．では，プログラムを作成しましょう．

```
class InputCommand
{
        public static void main(String[] args)
        {
            Sytem.out.println(”こんにちは” + args[0] + ”さん” );
        }
}
```

==========

実行するには，プログラム名の後にスペースを空けて自分の名前を打ち込んでからリターンキーを押す．

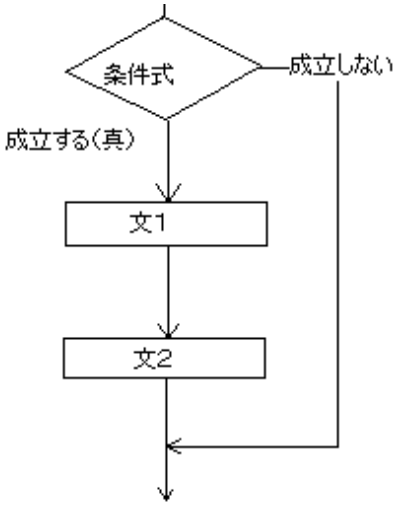
実行結果



練習問題 3.2　コマンドライン引数

args[0] と args[1] は第 1 引数と第 2 引数となります．そこで，2 つの引数 English と Math を用いて，”好き目は English and Math です”と表示するプログラムを作成しなさい．

if 文 (if Statement)　もし～ならば～する



if 文の記述法は次のようです．

if (条件式) 文 1 ;

ここで，条件式が真ならば，文 1 を実行する．

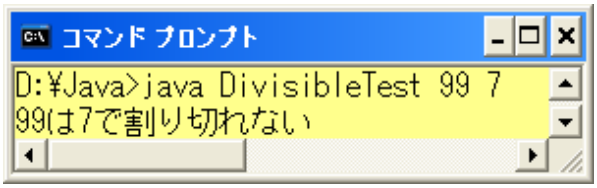
例題 3.4 整除可能性のテスト
2 つの整数をコマンドライン引数として用意し，1 つがもう 1 つの整数で割れるか判断するプログラムを作成しなさい．

解答 n%d は整数 n を整数 d で割ったときのあまりを表すので，n%d が零ならば割り切れ，そうでないならば割り切れないとなります．また，if 文の条件式は C/C++ と異なり，真または偽となるようにしなけらばなりません．そこで，if(n%d == 0) System.out.println(”割り切れる”); if(n%d != 0) System.out.println(”割り切れない”); と書くことができます．

コマンドライン引数は String 型ですので，これを整数に直して用いる必要があります．Java には**ラッパークラス (wrapper class)** とよばれる便利なものがあります．Integer 系ラッパークラスは，1 章で 10 進数の数を 16 進数に直すときに一度用いました．ここでは，数字からなる文字列を整数に直す Integer.parseInt() を用います．

```
class DivisibleTest
{
        public static void main(String[] args)
        {
            int first = Integer.parseInt(args[0]);
            int second = Integer.parseInt(args[1]);
            if(first%second == 0)
            {
                System.out.println(first + ”は” + second + ”で割り切れる”);
            }
            if(first%second != 0)
            {
                System.out.println(first + ”は” + second + ”で割り切れない”);
            }
        }
}
```

実行結果



練習問題 3.3 % の用い方

% と/を用いて整数 34 の並びを逆にするプログラムを作成しなさい.

if 文の後の中括弧 { と } は，セミコロン; で終了する文が一つの時は必要ありませんが，私の美的センスで書いています.

if … else 文 (if else Statement) もし～でなければ～する

例 3.1 式の場合

```
=======================================================
class IfElseStatement
{
        public static void main(String[] args)
        {
            int a = 0, b = 1;
            if(a*b == 0)
            {
                System.out.println(”a*b は 0 です”);
            }
            else
            {
                System.out.println(”a*b は 0 ではありません”);
            }
        }
}
=======================================================
```

実行結果

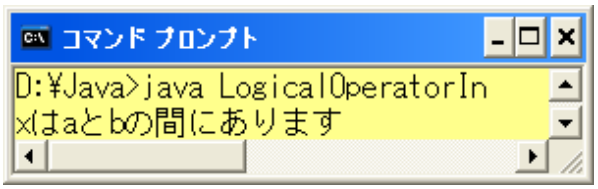
分岐条件の組み合わせ　－　論理演算子 (Logical Operators)

分岐の条件は範囲の指定などのときに，複数組み合わせられる．その組み合わせには，主に論理積＆＆と，論理和｜｜の演算が用いられる．

例 3.2 論理積（＆＆）の場合：x が a から b の間にある場合

```
=========================================================
class LogicalOperatorIn
{
        public static void main(String[] args)
        {
            int a = 1, b = 3, x = 2;
            if(a <= x && x <= b)
            {
                System.out.println(”x は a と b の間にあります”);
            }
            else
            {
                System.out.println(”x は a と b の間にはない”);
            }
        }
}
=========================================================
```

実行結果



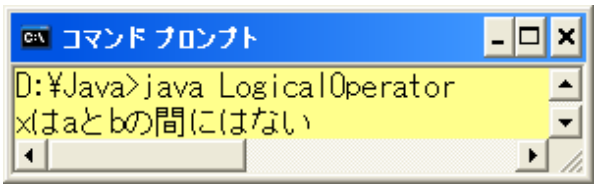
例 3.3 論理和（｜｜）の場合：x が a 以下または b 以上にある場合

```
=========================================================    class
LogicalOperator
{
        public static void main(String[] args)
        {
            int a = 1, b = 2, x = 3;
            if(a <= x || x >= b)
            {
                System.out.println(”x は a と b の間にはない”);
            }
            else
            {
                System.out.println(”x は a と b の間にはあります”);
```

```
        }
    }
}
```

==========

実行結果



単純な分岐 (Selection) －等価演算子，関係演算子 (Comparison Operator)

単純な分岐とは，条件式に 1 つの式だけを使った選択構造です．

おもに式としては，等価演算子（＝），関係演算子（＜ ＞）が使用されます．変数や式が単独で設定されることもあります．その場合は，その評価値が 0 ならば偽，0 以外ならば真という判定になります．

```
x < y     // x は y より小さい
x > y     // x は y より大きい
x <= y    // x は y 以下である
x >= y    // x は y 以上である
x == y    // x は y と等しい
x != y    // x は y と等しくない
```

論理演算子 (Logical Operators)

コンピュータの計算は論理演算を用いて行なわれる．基本的な論理演算には，**論理積 (AND)**，**論理和 (OR)**，**論理否定 (NOT)**，**排他的論理和 (EOR,XOR)** があります．なお，論理否定は単に否定ともいいます．

論理演算の結果をまとめたものを**真理値表 (truth table)** といい，次のような表です．

A	B	論理積	論理和	排他的論理和	論理否定
		A·B	A+B	A $\oplus$ B	$\bar{A}$
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

また，排他的論理和は，論理積，論理和，否定を用いて次のように展開できます．

$$A \oplus B = A \cdot \bar{B} + \bar{A} \cdot B$$

条件式には&&(AND), ||(OR), !(not) の 3 つの論理演算子を用いることができます．真理値表より，次の結果を得る．

DeMoivre の定理

1. !(A || B) = !A && !B
2. !(A && B) = !A || !B
3. A || (B && C) = (A || B) && (A || C)
4. A && (B || C) = (A && B) || (A && C)

if 文 (if Statement)：もし〜ならば〜する

例題 3.5　2 数の比較

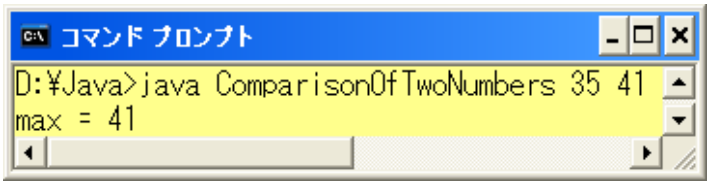
2 つの異なる数字を読み取って比較し，大きい方を求めるプログラムを（if 文だけを使って）作成しなさい．

解答

変数 a と b にコマンドライン引数を入れます．つぎに，a と b の値を比較をします．もし a が b より大きいならば max は a です．もし b が a より大きいならば max は b であるとなります．では，プログラムを作成しましょう．

```
==================================================
class ComparisonOfTwoNumbers
{
        public static void main(String[] args)
        {
            int a = Integer.parseInt(args[0]);
            int b = Integer.parseInt(args[1]);
            int max;
            if(a > b)
            {
                max = a;
            }
            else
            {
                max = b;
            }
            System.out.println("max = " + max);
        }
}
==================================================
```

実行結果



else-if 文 (else-if Construct)：もし～ならば～し，そうでなければ～し，それでもなければ～する

この構文は if-else の else に新たに条件が加わった形です．この構文は選択条件が 3 つ以上あっても対処できます．

練習問題 3.4 else-if 文の例　3 数の比較

3 つの数字 30, 10, 40 を読み取って比較し，一番大きい数字を求めるプログラムを作成しなさい．

例題 3.6 うるう年の判定

西暦年数 year をコマンドライン引数として入力し，うるう年かを判定するプログラムを作成しなさい．

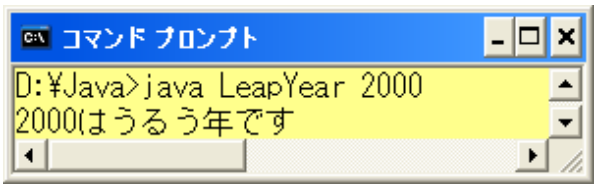
解答 うるう年とは 400 で割り切れる年数．または，4 で割り切れ，かつ 100 で割り切れない年数のことです．例えば 1900 年はうるう年ではありません．なぜならば，4 と 100 の両方で割り切れてしまうからです．

これをプログラムにするには，if (year % 400 == 0) ならばうるう年．else if (year % 4 == 0 && year % 100 != 0) ならばうるう年．else うる年ではないとすればよいでしょう．ではプログラムを作成しましょう．

```
=====================================================
class LeapYear
{
        public static void main(String[] args)
        {
            int year = Integer.parseInt(args[0]);
            if(year%400 == 0)
            {
                System.out.println(year + ”はうるう年です”);
            }
            else if(year % 4 == 0 && year % 100 != 0)
            {
                System.out.println(year + ”はうるう年です”);
            }
            else
            {
```

```
            System.out.println(year + "はうるう年ではありません");
        }
    }
}
```

実行結果



if(year%400 == 0) 以降は次のように書くこともできます．

if(year%400 == 0 || year%4 == 0 && year%100 != 0)

練習問題 3.5 うるう年の判定

if(year%400 == 0 || year%4 == 0 && year%100 != 0) を用いて，うるう年判定プログラムを作成しなさい

Switch 文 (switch Construct)

Java 言語には else-if 文の代わりに switch 文とよばれるものがあります．switch 文で書けるものはすべて else-if 文で書けます．しかし，逆は真ではありません．swith 文の構文は次のようになります．

```
switch (式)
{
    case 定数式 1 :
                        文 1
                        break;
    case 定数式 2 :
                        文 2
                        break;
    default :
                        文
                        break;
}
```

switch 文の動作

「ステップ 1 」

式の値を求める

「ステップ 2 」

式の値が

（1） case の定数式と等しい場合，

その case に続く文にプログラムの制御が移り実行され,break 文に出会うか，swi

（2） どの case とも等しくないが，default が存在するとき，

default に続く文にプログラムの制御が移り，実行されます．

（3）どの case とも等しくなく，default が存在しないとき，

プログラムの制御は，switch 文の次の文に移る．

各 case には 1 つ以上の数値を持つ定数あるいは定数式による名札を付ける．Default という名札の付いた case は，他の case のどれもが満足されなかったときに実行されます．Java の switch 文は，C/C++ の悪い習慣 (すり抜け) をそのまま引きずっており，break 文を省略することができます．しかし，すり抜けの機能はバグの温床になるので，使わないようにして下さい．

例題 3.7 順位の判定

コマンドラインから順位を入力し，入賞者の判別を行うプログラムを作成しなさい．

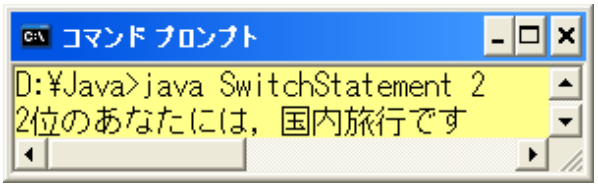
解答

```
=====================================================
class SwitchStatement
{
        public static void main(String[] args)
        {
            int rank = Integer.parseInt(args[0]);
            switch (rank)
            {
                case 1:
                    System.out.println(”優勝者には海外旅行です”);
                    break;
                case 2:
                    System.out.println(”2 位のあなたには，国内旅行です”);
                    break;
                case 3:
                    System.out.println(”3 位のあなたには，図書券です”);
                    break;
                default:
                    System.out.println(”残念賞のタオルをどうぞ”);
                    break;
            }
        }
```

```
}
```

==========

実行結果



三項演算子 (Ternary Operator)

Java には，if ... else 文の代わりに用いることができる演算子があります．その演算子は**三項演算子 (ternary operator)** と呼ばれ，その構文は次のようになります．

```
条件 ? ステートメント 1 : ステートメント 2;
```

ここで，条件が満たされると，ステートメント 1 が実行され，満たされない場合はステートメント 2 が実行されます．例えば，構文

```
min = (x < y ? x : y);
```

は，x と y の小さい方を min に代入します．

三項演算子は条件とステートメントが単純なときに用いるくらいで，複雑なステートメントの時は用いないようにします．

例題 3.8　2 数の比較

2 つの数字をコマンドライン引数として入力し，三項演算子を用いて大きい方を求めるプログラムを作成しなさい．

解答

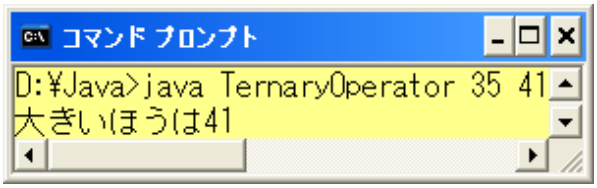
3 項演算子 (a > b ? a:b) により大きいほうを取り出せます．

==========

```
class TernaryOperator {
        public static void main(String[] args)
        {
            int a = Integer.parseInt(args[0]);
            int b = Integer.parseInt(args[1]);
            System.out.println("大きいほうは" + (a > b ? a : b));
        }
}
```

==========

実行結果



練習問題 3.6 3 数の最大値

3 つの数字を読み取って比較し，三項演算子を用いて一番大きな数字を求めるプログラムを作成しなさい．

スコープ (Scope)

識別子のスコープとは，その識別子を用いることができるプログラムの部分です．例えば，変数は宣言されるまで用いることはできません．よって，そのスコープはその変数が宣言されているところで始まる．

例題 3.9 変数のスコープ

```
============================================================
class LifeOfScope
{
        public static void main(String[] args)
        {
            x = 11; // エラー：これは x のスコープに入っていありません．
            int x;
            { x= 22; // OK:これは x のスコープに入っています．
                y= 33; // エラー：これは y のスコープに入っていありません．
                int y; // OK:これは y のスコープに入っています．
                x = 44; // OK:これは x のスコープに入っています．
                y = 55; // OK:これは y のスコープに入っています．
            }
            x = 66; // OK:これは x のスコープに入っています．
            y = 77; // エラー：これは y のスコープに入っていありません．
        }
}
============================================================
```

x のスコープは x が宣言されたところから main メソッドの最後までです．y のスコープは y が宣言されたところからブロックの終了までです． ■

確認問題 3.1

1. 次の命題は正しいか調べなさい.

a. !(p || q) と !p || !q は同値です.

b. !!!p と!p は同値です.

c. p && q || r は p && (q || r) と同値である.

2. 次の命題を表す論理式を作成しなさい.

a. score は 80 以上で 90 よりも低い

b. answer は'N' か'Y' のどちらかである

c. n は 3 で割り切れるが，30 では割り切れない

d. ch は大文字である

4. 次のプログラムコードはどこがいけないか.

```
if (x == 0)
    if (y == 0) System.out.println("x と y は共に 0 です" );
else System.out.println("x は 0 ではありません" );
```

5. 次の式の値を求めなさい.　(x < y ? -1: (x == y ? 0 : 1));

6. x の絶対値を absx に代入する文を 3 項演算子を用いて表しなさい.

演習問題 3.1

1. 代入を使って，変数 a と b の内容を壊さずに，入れ替えるプログラムを完成しましょう．ただし，変数 a と b 以外に作業用として変数 w を用いることとします.

```
=====================================================
class SwapNumbers
{
    public static void main(String[] args)
    {
        int a,b,w;
        a=10;
        b=20;
        System.out.println("a = " + a);
        System.out.println("b = " + b);
        //入れ替え
        (ア)
        (イ)
        (ウ)
        System.out.println("a=" + a);
        System.out.println("b=" + b);
    }
```

```
}
```

==========

2. 次のプログラムを実行した結果を数値で答えなさい.

==========

```
class DivAndRem
{
        public static void main(String[] args)
{
            int a,b;
            a=9;
            b=2;
            System.out.println(”a/b の結果= ” + (a/b)); /*整数どうしの割り算を行います. 結果は
            整数で表されます. */
            System.out.println(”a%b の結果=” + (a%b) ); // a%b は a を b で割ったあまりを出す//
}
```

==========

第 4 章

反復

反復 (Loop)

ループとは繰り返し文のことです．俗に反復処理 (iteration) と呼ばれていて構造化プログラミングの根幹をなしています．反復処理には while 文，do-while 文と for 文があります．

その昔，コンピュータの性能が悪かった時代，また，しっかりした制御構造および反復処理がプログラミング言語に用意されたいなっかた時代，多くの人が goto 文と呼ばれるジャンプを使ってプログラムを書いていました．しかし，１９６０年代後半 goto 文によって書かれたプログラムは，他人にとってわけのわからないプログラム（スパゲッティプログラム）になり，諸悪の根源であると言われ始めました．このことより，ソフトウェアも通常の工業製品と同じように生産管理が必要であるという観点から「ソフトウェア工学」が生まれました．このような流れの中から，分かりやすいプログラムを書くための方法論がダイクストラによって提案され，その中の 1 つに「構造化プログラミング」があります．

構造化プログラミング (Structured Programming)

構造化プログラミングは，基本的に 3 つの構造から成り立っています．

1. 連接 (sequence):プログラムは上から下に流れる
2. 選択構造（Selection）
3. 反復処理（Iteration）

構造化定理によると，すべてのアルゴリズムはこの 3 つの構造だけで記述できることが証明されています．

現在，Java 言語を学んでいる人は，上記の 1 ，2 ，3 だけでアルゴリズムを書く練習をして下さい．なお，今やプロジェクトの規模が大きくなり，構造化プログラミングを用いても，ある時点からは複雑さがプログラマの管理できる限界を超えてしまっています．そこで生まれたのが，オブジェクト指向という考え方です．Java 言語はすでに述べましたがオブジェクト指向言語です．

while 文 (while Statement)：〜の間〜する

while 文の構文は

```
while(条件式){
        ステートメント 1;
        ステートメント 2;
        ....;
}
```

の形をとります．while 文は条件式が真である限り，その対象の評価を繰り返します．条件式が偽になると，ループを停止します．したがって，while 文はステートメントを一つも実行せず，while 文を終えることもあります．

例題 4.1 数列の和
$1^2+2^2+3^2+4^2+5^2+\cdots+100^2$ を求めるプログラムを while ループを用いて作成しなさい．

解答 求める和を s とおくと，$s=\sum_{k=1}^{100} k^2$ と表せます．$k=1$ のとき，$s=1^2$．$k=2$ のとき，$s=1^2+2^2$ となりますが，$k=1$ のとき，$s=1^2$ としておくと，$k=2$ のとき，$s=s+2^2$ と表せることに気づいたでしょうか．これが数列の和のコーディングの方法です．

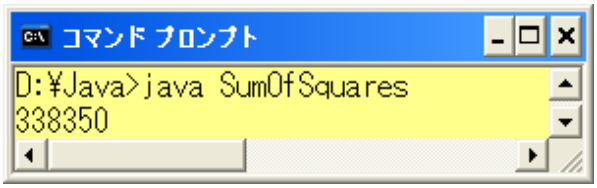
この問題では j 番目が j^2 より和は $s=s+j^2$ と表せます．これを j が 1 から 100 まで行えばよいので, 擬似コードで書くと，

```
j ← 1, s ← 0;
while (j <= 100) do
  s ← s + j^2;
  j ← j+1;
```

となります．これを Java 言語に書き直すと，

```
class SumOfSquares
{
        public static void main(String[] args)
        {
            int j = 1, sum = 0;
            while(j <= 100)
            {
                sum += j*j;
                j++;
            }
            System.out.println(sum);
        }
}
```

実行結果



一般に，j 番目がある関数 f（j）で表せるならば，上のステートメント s += j*j; を s += f(j) と書き換えることができます..

練習問題 4.1 数列の和

$100^2 + 99^2 + 98^2 + 97^2 + 96^2 + \cdots + 1^2$ をこの順で求めるプログラムを while ループを用いて作成しなさい

例題 4.2 逆数の和

s = 1 + 1/2 + 1/3 + ··· 1/n を求めるプログラムを作成しなさい．ただし，n は逆数の和 s が入力されたある整数より大きくなる最初の整数とします．

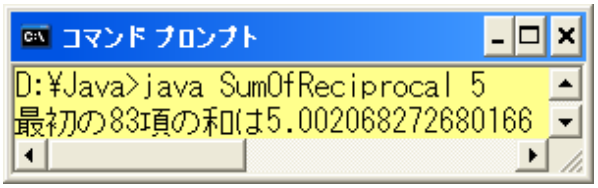
解答 逆数の和を sum とし，入力された値を bound とすると，

```
i ← 0;
while(sum < bound) do
    sum ← sum + 1.0/++i;
```

で逆数の和が入力された値より大きくなる最初の整数 i を求めることができます．Java 言語に書き直すと，

```
=================================================
class SumOfReciprocal
{
        public static void main(String[] args)
        {
            int bound = Integer.parseInt(args[0]);
            int i = 0;
            double sum = 0.0;
            while(sum < bound)
            {
                sum += 1.0/++i;
            }
            System.out.println("最初の" + i + "項の逆数の和は" + sum);
        }
}
=================================================
```

実行結果



練習問題 4.2 逆数の和

sum = 1 + 1/2 + 1/3 + ··· + 1/1000 と rsum = 1/1000 + 1/999 + ··· + 1 を float で宣言し求めるプログラムを作成しなさい．sum と rsum の値を比較しなさい．また，どちらが正確か考えなさい．さらに，このとき発生した誤差はなんとよばれるか．

例題 4.3　素因数分解

2 以上の整数をコマンドラインから入力して，素因数分解するプログラムを作成しなさい．

解答 素因数とは，与えられた数 n の約数でかつ素数のことです．素数とは，それ自身と 1 でしか割れない数です．ただし，1 は素数には入れません．例えば，n を 120 とすると，n の約数は，2,3,5 であり，120 の素因数分解を行なうと，$120 = 2\ 3 \cdot 3 \cdot 5$ となります．

2	120
2	60
2	30
3	15
	5

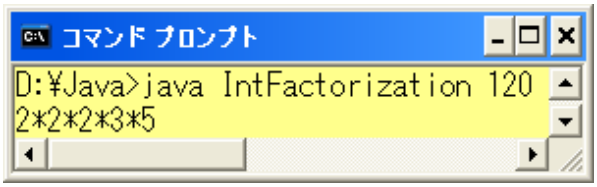
では，素因数分解をおこなうアルゴリズムを考えてみます．データ n を読み取って，2 から順に割っていき，ある数 m で割れたら商 n/m を新たな n とおき，さらに m で割り切れなくなるまで続ける．m で割り切れなくなったら m を 1 増加し，n を割っていくことを繰り返し，商 n が 1 になるまで続ける．このとき発生した，m が素因数となります．表示するときに，最後の数字のあとには「,」を付けないようにします．では，プログラムを作成しましょう．

```
==========================================================
class IntFactorization
{
        public static void main(String[] args)
        {
            int input = Integer.parseInt(args[0]);
            int divisor = 2;
            while(input != 1)
            {
                while(input % divisor == 0)
                {
```

```
                input /= divisor;
                if(input != 1)
                {
                    System.out.print(divisor + "*");
                }
                else
                {
                    System.out.println(divisor);
                }
            }
            divisor++;
        }
    }
}
```

==========

実行結果



練習問題 4.3 素数判定

与えられた数が素数かどうか判定するプログラムを作成しなさい.

do ... while 文 (do ... while Statement)

do ... while 文の構文は

```
do {
    ステートメント;
}while(条件);
```

の形をとります．ここで，条件式は真または偽をとります．条件が真でなくなるまでステートメントは実行されます．つまり，最低一回は必ず実行されます.

例題 4.4 数列の和

$1^2 + 2^2 + 3^2 + 4^2 + 5^2 + \cdots + 100^2$ を求めるプログラムを do...while ループを用いて作成しなさい.

解答

```
=====================================================
class SumOfSquares
{
        public static void main(String[] args)
        {
            int j = 1, sum = 0;
            while(j <= 100)
            {
                sum += j*j;
                j++;
            }
            System.out.println(sum);
        }
}
=====================================================
```

実行結果

練習問題 4.4 数列の和

$100^2 + 99^2 + 98^2 + 97^2 + \cdots + 1^2$ を求めるプログラムを do...while ループを用いて作成しなさい．

階乗 0!,1!,2!,3!,... は次の漸化式により定義されています．

$$\begin{cases} 0! = 1 \\ n! = n(n-1)! \end{cases}$$

例えば，$n = 3$ のとき，

$$3! = 3(3-1)! = 3(2!) = 3(2)(2-1)! = 3(2) = 6$$

となるので，6 は 3 の階乗です．

例題 4.5 階乗数

入力された整数までの階乗数を全て求めるプログラムを作成せよ．

解答 階乗 $n!$ を $f(n)$ とすると，$f(n) = nf(n-1)$ で与えられる．そこで，$f = f(n-1) = (n-1)!$ とおくと，階乗を求める構文は，繰り返しを用いて

```
f = f * n
```

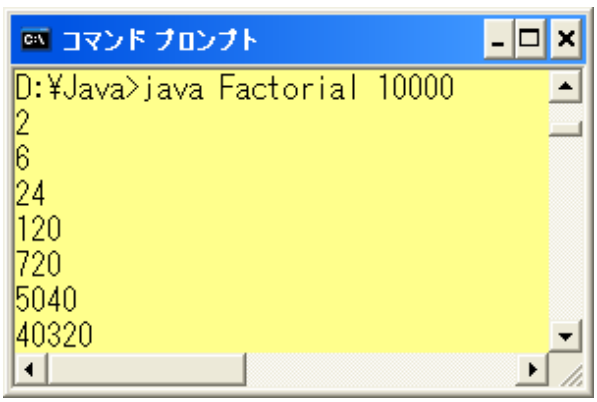
となる．この n を 1 から順に増やしていけば，全ての階乗を求めることができる．よって，擬似コードで書くと

```
入力 bound;
i ← 1;
f ← 1;
do
    f ← f * (++i);
while(i < 入力);
```

となる．例題 5.2 では**再帰 (recursion)** を用いて n!を求めている．

```
=================================================
class Factorial
{
        public static void main(String[] args)
        {
            long input = Integer.parseInt(args[0]);
            long f = 1, i= 1;
            do
            {
                f *= ++i;
            System.out.println(f);
            }while(f < input);
        }
}

=================================================
```

実行結果



練習問題 4.5 階乗

${}_nC_r$ を求めるプログラムを作成しなさい．ここで，

$$ {}_nC_r = 1 \cdot \frac{n-1+1}{1} \cdot \frac{n-2+1}{2} \cdot \frac{n-3+1}{3} \cdots \frac{n-r+1}{r} $$

for 文 (for Statement)

for 文の構文は

```
for (初期値；条件；更新){
        ステートメント;
}
```

の形をとります．

1. 初期値はコントロール変数の初期化を行います．2 つ以上の初期値を書く場合はカンマで区切る．
2. 条件が成立するとステートメントは実行されます．条件が成立しなければループを終了．
3. コントロール変数が更新されます．
4. 2,3 を繰り返す．

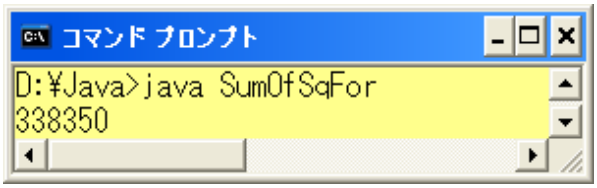
例題 4.6 数列の和

$1^2 + 2^2 + 3^2 + 4^2 + 5^2 + \cdots + 100^2$ を求めるプログラムを for ループを用いて作成しなさい．

解答

```
================================================
class SumOfSqFor
{
        public static void main(String[] args)
        {
            int sum = 0;
            for(int j = 1; j <= 100; j++)
            {
                sum = sum + j*j;
            }
            System.out.println(sum);
        }
}

================================================
```

実行結果



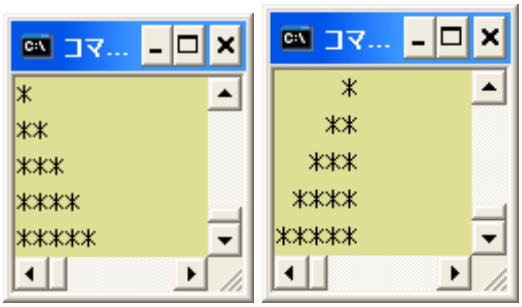
練習問題 4.6 for ループを用いた数列の和

$1/1^2 + 1/3^2 + 1/5^2 + \cdots + 1/101^2$ を求めるプログラムを for ループを用いて作成しなさい．

例題 4.7 反復練習

次のような形を表示するプログラムを作成しなさい．

実行結果



解答

＜実行結果 1 ＞

System.out.println(”*”);

で 1 つのアスタリスクを表示することができます．では*****と表示するにはどうすればよいでしょうか．確かに System.out.println(”*****”); でもできますが，これではアスタリスクを任意の数だけ表示せよとなると，不可能です．そこで，System.out.print(”*”); を 5 回実行すると*****が表示できることに気づけばよいでしょう． 5 回実行させるには，例えば for ループを用いると for(int i=1;i<=5;i++) と書けばよいことを知っています．次に，第 1 回目には， 1 つのアスタリスク，第 2 回目は 2 つのアスタリスクと増やすにはどうすればよいでしょうか．i<=5 の 5 の代わりに，j を用いて，

```
for(int j=1;j<=5;j++){
    for(int i=1;i<=j;i++){
        System.out.print(”*”);
    }
}
```

と書けば，外側のループが 1 回終了する間に内側のループが j 回実行されることになります．内側のループが終了したら，”
n”で改行します．ではプログラムを作成しましょう．

```
================================================
class AstTree
{
        public static void main(String[] args)
        {
            for(int j = 1; j <= 5; j++)
            {
                for(int i = 1; i <= j; i++)
                {
                    System.out.print(”*”);
                }
                System.out.println();
            }
        }
}
================================================
```

＜実行結果 2＞

実行結果 1 との違いはアスタリスクを右詰めで表示しているところです．そこで，アスタリスクの前にブランクを挿入するコマンドを追加すればよいでしょう．

```
================================================
class AstTreeRight
{
        public static void main(String[] args)
        {
            for(int j = 1; j <= 5; j++)
            {
                for(int i = 1; i <= 5; i++)
                {
                    if(i > 5-j)
                    {
                        System.out.print(”*”);
                    }
                    else
                    {
                        System.out.print(” ”);
                    }
                }
                System.out.println();
            }
        }
```

```
}
================================================
```

練習問題 4.7 反復練習

次のような形を表示するプログラムを作成しなさい．

continue 文 (continue Statement)

break 文はループブロックの break 文以下をスキップし，ループの外の次の文にジャンプします．continue 文も break 文のように，ループブロックをスキップしますが，ループを終了する代わりに，ループの次の反復に移ります．

例題 4.8 continue と break 文

次のプログラムを実行するとどんな結果が得られるか考えなさい．

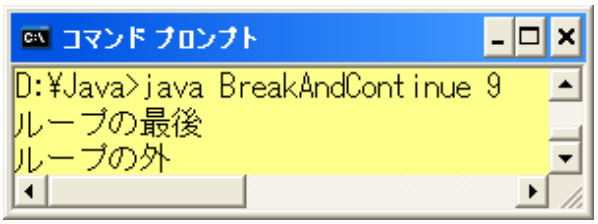
```
================================================
class BreakAndContinue
{
        public static void main(String[] args)
        {
            int n = Integer.parseInt(args[0]);
            for(int i = 1; i < n ; i++)
            {
                if(n % 2 == 0)
                {
                    System.out.println("ループの中");
                    continue;
                }
                if(n % 3 == 0)
                {
                    System.out.println("ループの最後");
                    break;
                }
            }
```

```
            System.out.println(”ループの外”);
        }
}
```

==========

解答 2 でも 3 でも割り切れない数を入力すると，2 つの if 文の条件を満たさないので，”ループの外”が出力されます．2 で割り切れる数を入力すると，”ループの中”が出力され continue で残りのステートメントをスキップし，ループに戻り，ループの条件が満たされなくなったら”ループの外”を出力します．2 で割り切れず 3 で割り切れる数を入力すると，”ループの最後”を出力し break 文でループの外に出ます．

実行結果



確認問題 4.1

1. while 文の条件式が最初から正しくない（つまり零）場合どうなるか述べなさい．

2. 次のプログラムコードはどこが間違っているか．

```
while (n <= 100)
sum += n*n;
```

3. 次のプログラムコードはどこが間違っているか．

```
class WhatIsWrong
{
        public static void main()
        {
            final double PI;
            int n;
            PI = 3.141592653;
            n = 22;
        }
}
```

4. **break** 文はどのようにしてループにコントロールをもたらすか述べなさい．

演習問題 4.1

1. e を式，s を文とするとき，次の **for** ループを **while** ループを用いて書き直しなさい．

a. for(;e;)s

b. for (; ; e) s

2. 次の **for** ループを **while** ループを用いて書き直しなさい．

```
for (int i=1; i <= n; i++)
        System.out.println(i*i + " ");
```

3. 次のコードをトレースしなさい．

```
float x = 4.15;
for (int i = 0; i < 3; i++)
   x *= 2;
```

4. 入力した 10 進数の数字を反転して出力するプログラムを書きなさい．

5. Euclid の互助法は 2 つの整数の最大公約数を求めるのに用いることができます．そのアルゴ

リズムは例えば，210 と 56 の最大公約数ならば，

$$210 = 56 * 3 + 42 \cdots (1)$$
$$56 = 42 * 1 + 14 \cdots (2)$$
$$42 = 14 * 3 + 0 \cdots (3)$$

より，14 は（3）の式の両辺の約数です．次に 14 は（2）の式の両辺の約数です．なぜなら，14 は 42 の約数より，14 は 56 の約数となるからです．最後に 14 は（1）の式の両辺の約数です．つまり，14 は 210 と 56 の約数です．また，14 より大きな数で，210 と 56 の約数があるとすると，その数は（1）の式より 42 の約数となります．ということは（2）の式より，14 の約数となります．このことから，14 は 210 と 56 の最大公約数です．

次に，210 と 56 の最大公約数を $(210, 56)$ と表すと，

$$(210, 56) \to (56, 42) \to (42, 14) \to (14, 0)$$

となります．このアルゴリズムを実装しなさい．

第 5 章

メソッド (method)

あるまとまった処理をするためにいくつかの命令を組み合わせた小さなプログラムをメソッドといいます．Java では実行文はすべて，クラスの中に記述されるメソッドに含まれていなければなりません．

メソッドの基本 (Basics of Methods)

メソッドは，ヘッドとボディでできていて，

[修飾子] 戻り値のデータ型　メソッド名 (データ型　仮引数１，データ型，仮引数２，)

```
{
        実行文;
        return(式);
}
```

の形をとります．例えば，

```
=====================================================
public static int cube(int x)
{
        return x*x*x;
}
=====================================================
```

において，public static int cube(int x) がヘッドで，ボディは { return x*x*x; } です．

メソッドは値を返すという性質を持っていますので，そのメソッドがどのような型のデータを返すかをメソッド名の前に，メソッドの戻り値のデータ型として宣言する必要があります．return（式文）の式文の値が関数の返す値として呼び出し元に返されるので，式の値と関数のデータ型は一致しなければなりません．値を戻さないメソッドの場合は，メソッドのデータ型として void 型を指定し，式の記述のない return 文を指定します．また，値を返さないメソッドの場合，return 文自体を省略することができます．

この例はメソッドの中でも最も単純なものの一つです．一般にメソッドのボディはもっと大きいが，メソッドのヘッドはほとんどの場合一行で収まります．

main() もメソッドで，そのヘッドは

　　public static void main(String[] args)

です．そのボディはプログラム自身です．main() の戻り値の型は void であり，関数名は main です．

メソッドの return 文はメソッドの実行を終了する役目とメソッドを呼んだプログラムに値を戻す役目があります．そして，その構文は

　　return 式文;

となります．

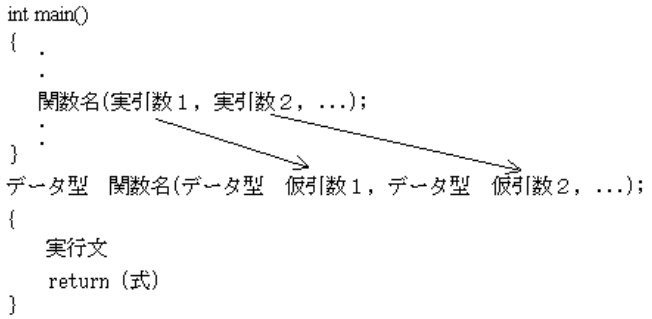
メソッドの種類

Java のメソッドはインスタンスメソッドとクラスメソッドに分けることができます．インスタンスメソッドは普通に記述するメソッドです．一方のクラスメソッドは static 修飾子を付けますので，これを static メソッドとよぶこともあります．両者はメソッドの指定方法に違いがあります．

クラスメソッドは，クラス名．クラスメソッド名 ();
インスタンスメソッドは，インスタンス名．インスタンスメソッド名 ();
のようにして，指定します．

メソッドの呼び出し (Invoking Methods)

呼び出し側と呼び出されるメソッドの間でのデータの受け渡しは引数を用いて行なわれます．呼び出し側で使用する引数を**実引数 (actual parameter)**，メソッド側で使用する引数を**仮引数 (formal parameter)** といいます．メソッドの定義において，引数のデータ型を宣言する必要があります．実引数には，定数，変数，式を記述することができますが，仮引数はデータを受け取る器として使用されるので，記述できるのは変数だけです．また，実引数と仮引数のデータ型は必ず一致していなければなりません．



基本データ型の受け渡し (値渡し法 (Pass by Value))

Java で引数 (実引数) の値をメソッド (仮引数) へ渡す方法を**値渡し (pass by value)** といいます．メソッドが呼び出されたとき，実引数の値は仮引数のメモリ上にコピーされ，メソッド側

で仮引数にコピーされた値を変更しても，呼び出し側の実引数の値には影響が及ばない方法です．基本データ型は全てこの方法で渡されます．値渡しによるデータの受け渡しは，呼出し側から呼び出されるメソッド側への一方通行となります．

例題 5.1 データを読み込んで和と差を求める
main メソッドから実数データ a,b をユーザメソッド sum とユーザメソッド diff に渡して，その和と差を計算して表示するプログラムを作成せよ．

解答

1．2つの値 x,y の和を計算するユーザメソッドを sum()，差を計算するユーザメソッドを diff() とすると，ユーザメソッド sum() と diff() は次のように表せる．

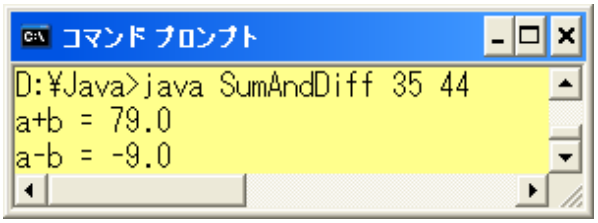
```
private static void sum(double a, double b){
    System.out.println("a+b = " + (a+b));
}
```

```
private static void diff(double a, double b){
    System.out,println("a-b = " + (a-b));
}
```

2．main メソッドから読み込んだ a,b の値をユーザメソッド sum() に渡す．同様にユーザメソッド diff() に渡す．ではプログラムを作成しよう．

```
class SumAndDiff
{
        public static void main(String[] args)
        {
            double x = Double.parseDouble(args[0]);
            double y = Double.parseDouble(args[1]);
            sum(x,y);
            diff(x,y);
        }
        private static void sum(double a, double b)
        {
            System.out.println("a+b = " + (a+b));
        }
        private static void diff(double a, double b)
        {
            System.out.println("a-b = " + (a-b));
        }
}
```

実行結果



練習問題 5.1 平均を求める

main 関数から実数データ a,b を関数 ave に渡して，平均を計算して表示するプログラムを作成せよ．

参照型渡し（Pass by Reference）

Java では基本データ型以外は，参照型渡しが用いられます．参照型というと難しそうですが，通常は，基本データ型と同じように扱うことができます．

ローカル変数とメソッド (Local Variables and Methods)

ローカル変数はブロック内で宣言された変数です．したがって，そのブロック内からしかアクセスできません．メソッド自身，一つのブロックですので，メソッド内で宣言された変数はローカル変数です．また，メソッドの引数もローカル変数とみなされます．

例題 5.2 階乗関数

n!の計算をするユーザメソッド fact() を作成しなさい．

階乗は例題 4.5 で紹介した．n の階乗 $n!$ は n に n 以下の全ての正の整数を掛けたものです．

$$n! = n(n-1)(n-2)\cdots 3\cdot 2\cdot 1$$

```
==========================================================
private static int fact(int n)
{
        int f = 1;
        while (n > 1)
            f *= n--;
        return f;
}
==========================================================
```

実行結果

このメソッドは 2 つのローカル変数 n と f を持っています．引数 n はユーザメソッド fact() で宣言されているのでローカル変数です．変数 f はメソッドのボディで宣言されているので，ローカル変数です．

メソッドのオーバーロード (多重定義)(Overloading methods)

メソッドのオーバーロードとは，引数の型や数の違いによって，同じ名前のメソッドを用いることです．メソッドをオーバーロードすると，関連する複数の操作を同じ名前で参照できるので，プログラムの複雑さを軽減することが可能となります．

例えば，C のライブラリには，整数の絶対値を求めるための abs() 関数，長整数の絶対値を求めるための labs() 関数，そして，浮動小数点数の絶対値を求めるための fabs() 関数と 3 つの関数が含まれています．

しかし，これらの 3 つの関数はみな絶対値を返し，違いはデータ型だけです．このようなとき，Java では，3 つの異なるデータ型に対して，次のようにして 1 つの名前をオーバーロードすることができます．

```
int myabs(int n);
long myabs(long n);
double myabs(double n);
```

例題 5.3 myabs() の多重定義を行い，-10 の絶対値，-10.34 の絶対値を出力するプログラムを作成しなさい．

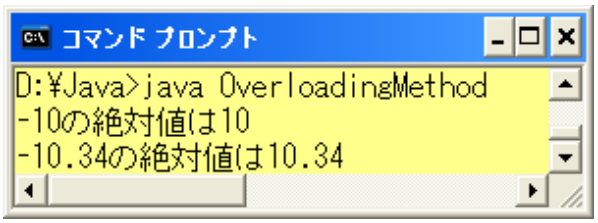
解答

```
=================================================
class OverloadingMethod
{
        public static void main(String[] args)
        {
            System.out.println(”-10 の絶対値は” + myabs(-10));
            System.out.println(”-10.34 の絶対値は” + myabs(-10.34));
        }
        private static int myabs(int n)
        {
            return n < 0 ? -n : n;
```

```
        }
        private static double myabs(double d)
        {
            return d < 0 ? -d : d;
        }
}
```

==========

実行結果



確認問題 5.1

1. 関数の宣言はどこにおかれるべきか

3. 関数の定義を別のファイルに置くことの利点を述べよ.

4. 引数による受け渡しと参照による受け渡しの違いを述べよ.

5. 次の宣言の間違いを直せ.

int f(int a, int b=0, int c);

演習問題 5.1

1. 次のうるう年を判定するプログラムコードはどちらが効率がよいか調べよ.

y % 4 == 0 && y % 100 != 0 || y % 400 == 0

y % 4 == 0 && (y % 100 != 0 || y % 400 == 0)

2. 4 つの整数から最も小さい整数を見つける次の関数を完成しなさい.

int min(int, int, int, int)

3. n 個から k 個を選び順番をつけて並べる並べ方 P(n,k) は

$$P(n,k) = \frac{n!}{(n-k)!}$$

で与えられる. この関数のプログラムコードを作成せよ.

4. 例題 5-3 のように, min(int, int) と min(double, double) の多重定義関数を作成し, min(10,20) と min(10.34, 20,45) を出力せよ.

第 6 章

クラス（class）

コンストラクタ (Constructors)

Triangle クラスは assign() メソッドを用いて，そのオブジェクトの初期化を行っています．しかし，オブジェクトが宣言されたときに，初期化するほうがより自然です．そこで，Java ではクラスオブジェクトに対して，初期化が行えるように**コンストラクタメソッド**を用意しています．

コンストラクタは特殊なメソッドで，オブジェクトが宣言されると自動的に呼び出されます．つまり，Triangle t = new Triangle() で，Triangle クラスのオブジェクト t が作成されますが，実はコンストラクタが自動的に呼び出されたのです．コンストラクタは，**クラスのオブジェクトを作る関数です．そのため，コンストラクタはクラスと同じ名前を取ります．**

例題 6.1 三角形クラス
底辺 18，高さ 24.5 の三角形の面積を求めるプログラムをコンストラクタを用いて作成しなさい．

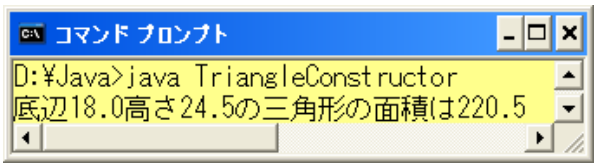
解答

1. Triangle クラスの宣言を行います．
2. コンストラクタの定義を行います．コンストラクタの定義は
   public コンストラクタ名 (型　引数)
   {
   }
   で行います．
3. メソッドの定義を行います．メソッドの定義は
   戻り値のデータ型　メソッド名 ()
   {
   }
   で行います．

4. 最後に main メソッドで Triangle クラス型のオブジェクトを作り，三角形の面積を求めます．

```
==================================================
class TriangleConstructor
{
        private double base, height;
        public TriangleConstructor(double b, double h)
        {
            base = b;
            height = h;
        }
        public double area()
        {
            return base*height/2;
        }
        public void display()
        {
            System.out.println("底辺" + base + "高さ" + height + "の三角形の面積は" + area());
        }
        public static void main(String[] args)
        {
            Triangle t = new Triangle(18,24.5);
            t.display();
        }
}

==================================================
```

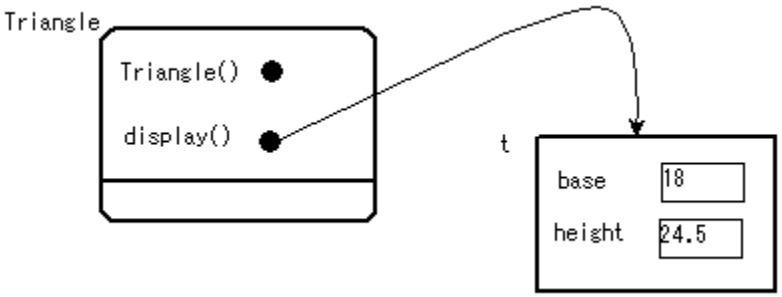
実行結果



Triangle t = new Triangle(18,24.5) により，base と height を持つオブジェクト t が作成され，18 と 24.5 がそれぞれ base と height に渡されます．

練習問題 6.1　上の例題で，底辺が 10，高さが 20 の三角形オブジェクト obj2 をコンストラクタを用いて作成その面積を求めなさい．

ここで，Triangle クラスとそのインスタンス化されたオブジェクトの関係を図で表すと次のようになります．



ここで，クラスは丸角矩形で表されメソッドを含んでいます．それぞれのメソッドは this ポインタを保持しています．this ポインタは，呼んでいるオブジェクトを指しています．この図は，上のプログラムの最後の行 t.display() で何が起きているのかを表しています．つまり，オブジェクト t が display() メソッドを呼び出しています．その瞬間，コンストラクタの this ポインタは呼ばれていないので **NULL** です．

クラスは 1 つ以上のコンストラクタを持つことができます．

メソッドへのオブジェクトの引渡し

メソッドに引数としてデータを渡すのと同じように，オブジェクトを引数として渡すことができます．それには，渡すオブジェクトの型を使用してメソッドの仮引数を宣言します．

例題 6.2 三角形クラス
底辺 18，高さ 24.5 の三角形の面積とメソッドを利用して面積の 2 乗を求めるプログラムをコンストラクタを用いて作成しなさい．

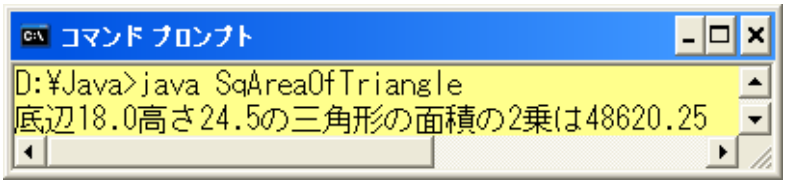
解答 面積の 2 乗を求める関数を double sqArea(); とします．

```
================================================
class SqAreaOfTriangle
{
        private double base, height;
        public SqAreaOfTriangle(double b, double h)
        {
            base = b;
            height = h;
        }
        public double sqArea()
        {
            double a = base*height/2;
            return a*a;
        }
        public void display()
```

```
        {
            System.out.println(”底辺” + base + ”高さ” + height + ”の三角形の面積の 2 乗は” +
sqArea());
        }
        public static void main(String[] args)
        {
            SqAreaOfTriangle t = new SqAreaOfTriangle(18,24.5);
            t.display();
        }
}
```

==========

実行結果



クラス変数とクラスメソッド

すでに述べたように，クラスは「フィールド」と「メソッド」といういう 2 種類のメンバを持っています．このフィールドとメソッドは厳密にはそれぞれ二つの性格のものに分けられ，計 4 つに分類されます．

ここでは，特殊なフィールド変数について学びます．Triangle クラスのオブジェクトを作成すると，それぞれのオブジェクトの base や height に値を代入したり出力したりすることができます．つまり，各オブジェクトごとにフィールド変数に値を格納することができます．このことを，フィールド変数 base, height は**オブジェクトに関連付けられている**といい，このフィールド変数を**インスタンス変数**といいます．また，display() メソッドも，オブジェクトを作成することによって，呼び出すことができます．このメソッドもオブジェクトに関連付けられているといい，このメソッドを**インスタンスメソッド**といいます．このように通常は，メンバはオブジェクトに関連付けられています．

ところが，クラスはオブジェクトに関連付けられていないメンバを持つことができます．これを**クラス全体に関連付けられている**といいます．クラスに関連付けられているフィールド変数を**クラス変数 (静的変数)** といい，クラスに関連付けられているメソッドを**クラスメソッド**といいます．

クラス変数は**スタティック (static)** という記憶クラス指定子を付けて宣言します．その構文は次のようになります．

```
class StaticX
{
```

```
        private static int n; // n をクラス変数として宣言
    }
```

これにより，クラス変数には，次のような 2 通りのアクセスの仕方が可能になります．

```
StaticX obj = new StaticX();
obj.n = 100;
```

または，

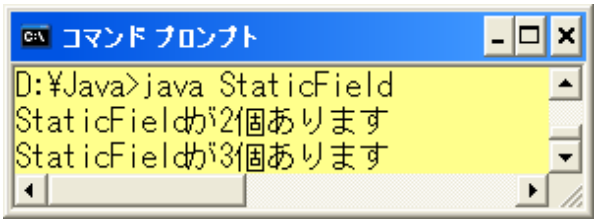
```
StaticX.n = 100;
```

例題 6.3 クラス変数

Triangle クラスに Triangle オブジェクトの数を数えるためのクラス変数 count を加える．Triangle オブジェクトが生成されるとカウンタは増え，Triangle オブジェクトが消滅するとカウンタは減る．

```
================================================
class StaticField
{
    private static int count = 0;
    public StaticField()
    {
        ++count;
    }
    public static void main(String[] args)
    {
        StaticField t = new StaticField();
        StaticField u = new StaticField();
        System.out.println("三角形が" + t.count + "個あります");
        StaticField v = new StaticField();
        System.out.println("三角形が" + t.count + "個あります");
    }
}
================================================
```

実行結果



クラスメソッド (Class Methods)

クラス変数が，クラス名．フィールド変数の形で用いることができたように，メソッドにも，クラス名．メソッド名 () の形で用いたほうが便利なものがあります．例えば，数学関係の処理を行う sin() メソッドがインスタンスメソッドである場合，sin() メソッドを呼び出すには，Math クラスのオブジェクトを次のようにして生成しなければなりません．

```
Math m = new Math();
double d = m.sin(1.2);
```

この手続きは面倒です．というのも生成された Math オブジェクト m は，sin() の計算に何の関与もしません．sin() メソッドに必要なデータはパラメターとして渡され計算結果はメソッドの戻り値として呼び出し元に返されます．クラスのインスタンスを作成したとしても，メソッドの呼び出し以外まったく必要のないものです．これを解決するため，static キーワードを使用して，メソッドをクラスメソッドとして定義します．

クラスメソッドを呼び出すには，

クラス名．メソッド名 (引数リスト);

の形をとります．

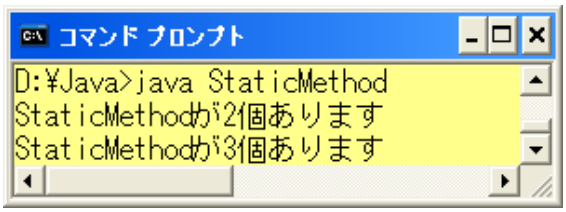
例題 6.4　クラスメソッド

クラスメソッドを用いて上の例題を書き直せ．

```
================================================
class StaticMethod
{
        private static int count = 0;
        public StaticMethod()
        {
            ++count;
        }
        private static int num()
        {
            return count;
        }
        public static void main(String[] args)
        {
            StaticMethod t = new StaticMethod();
            StaticMethod u = new StaticMethod();
            System.out.println("StaticMethod が" + StaticMethod.num() + "個あります");
            StaticMethod v = new StaticMethod();
            System.out.println("StaticMethod が" + StaticMethod.num() + "個あります");
        }
}
```

==========

実行結果

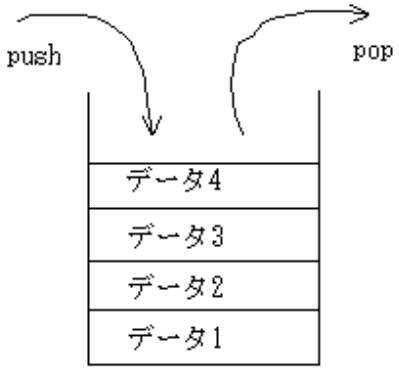


確認問題 6.1

1. クラスの公開メンバと非公開メンバとの違いを説明しなさい.
2. クラスメソッドとインスタンスメソッドとの違いを説明しなさい.
3. ディフォルトコンストラクタとそれ以外のコンストラクタとの違いを説明しなさい.
4. コンストラクタにはどんな名前を付けることができるか.
6. 1 つのクラスには何個のコンストラクタとディストラクタを含むことができるか.

演習問題 6.1

1. 空間上の点 (x,y,z) からなる Point クラスを作成しなさい．ただし，このクラスは，ディフォルトコンストラクタ，点の座標を負にする negate() 関数，距離を表す norm() 関数，そして表示のための print() 関数を含むとします.
2. 整数のスタックの Stack クラスを作成しなさい．ディフォルトコンストラクタ，ディストラクタ，push(), pop(), isEmpty(), isFull() 関数を含むものとします.



第 7 章

クラスの継承（Inheritance)

サブクラス (Subclass)

継承は，オブジェクト指向プログラミング (OOP) の基本の 1 つです．すでにあるクラスを用いて新しいクラスを作成したい場合があります．新しいクラスを最初から作成するのではなく，モデルとなるクラスを土台にして新しいクラスを記述できれば便利です．Java では，**extends** キーワードを用いて継承を行います．継承の元になるクラスを**スーパークラス (基本クラス)** といい，新しいクラスを**サブクラス (派生クラス)** といいます．

継承 (Inheritance)

派生クラスを作る方法はしばしば **is-a** の関係と言われます．なぜなら，”is”で定義された全てのオブジェクトは派生されたクラスのオブジェクトとなるからです．

クラスXからクラスYを派生する記述法は，

```
class Y extends X
{
        // 派生クラスに追加するメンバの宣言
}
```

です．ここで，Xはスーパークラス，Yはサブクラス．

例題 7.1 Person クラス
名前，国籍を持ち，名前と国籍を表示できる機能を持った人物クラス Person を作成しなさい．

解答 Person クラスにメインメソッドを含んでいる場合．

1. Person クラスの宣言を行なう．Person クラスは，名前，国籍を持つので，フィールド変数は，
   ```
   private String name,nationality;
   ```
   とする．
2. コンストラクタの定義を行なう．

```
    public Person(String name, String nationality)
    {
        this.name = name;
        this.nationality = nationality;
    }
```

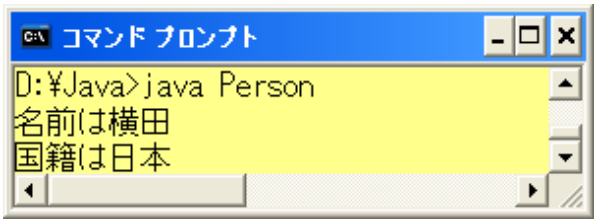
this.name はフィールド変数 name を指します．したがって，this.name = name; でコンストラクタが受け取った変数 name を用いて，フィールド変数の初期化を行います．

3. main メソッドの定義を行なう．

4. Person 型のオブジェクトを作り利用する．

```
==========================================================
class Person
{
    private String name, nationality;
    public Person(String name, String nationality)
    {
        this.name = name;
        this.nationality = nationality;
    }
    public static void main(String[] args)
    {
        Person p = new Person("横田","日本");
        System.out.println("名前は" + p.name);
        System.out.println("国籍は" + p.nationality);
    }
}

==========================================================
```

実行結果



解答 Person クラスにメインメソッドを含んでいない場合．

1. Person クラスの宣言を行なう．
   Person クラスは別のクラスで用いられるので，フィールド変数は，

protected String name,nationality;

とします．

2. コンストラクタの定義を行なう．

```
public Person(String name, String nationality)
{
    this.name = name;
    this.nationality = nationality;
}
```

3. 別のクラスを用意してメインメソッドの定義を行なう．この場合，ファイル名はメインメソッドのあるクラス名とする．

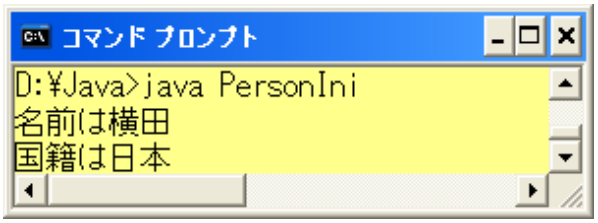
4. Person 型のオブジェクトを作り利用する．

==========

```
class Person
{
        private String name, nationality;
        public Person(String name, String nationality)
        {
            this.name = name;
            this.nationality = nationality;
        }
}
public class PersonIni
{
        public static void main(String[] args)
        {
            Person p = new Person(”横田”,”日本”);
            System.out.println(”名前は” + p.name);
            System.out.println(”国籍は” + p.nationality);
        }
}
```

==========

実行結果



スーパーコンストラクタの呼び出し

例題 7.2 クラスの拡張

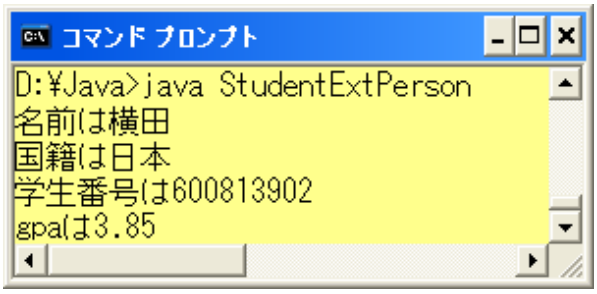
学生番号，成績評価をメンバ変数とする Student クラスを Person クラスのサブクラスとして作成せよ．

```
==========================================================
class Person
{
        protected String name, nationality;
        public Person(String name, String nationality)
        {
            this.name = name;
            this.nationality = nationality;
        }
}

  class Student extends Person
{
        protected String id;
        protected double gpa;
        public Student(String id, double gpa)
        {
            super(”横田”,”日本”);// スーパーコンストラクタの呼び出し
            this.id = id;
            this.gpa = gpa;
        }
}
public class StudentExtPerson
{
        public static void main(String[] args)
        {
            Student st = new Student(”600813902”,3.85);
            System.out.println(”名前は” + st.name);// st.name で Person クラスの name にアクセス
            System.out.println(”国籍は” + st.nationality);
            System.out.println(”学生番号は” + st.id);
```

```
            System.out.println("gpa は" + st.gpa);
        }
}
```

==========

実行結果



メンバへのアクセス

private メンバはクラスの中からだけアクセス可能です.
protected メンバはクラス中からとその派生クラスからもアクセス可能です.
public メンバはファイル内のどこからでもアクセス可能です.

一般に，protected はそのクラスの派生クラスを定義する可能性があるときに private の代わりに用います．派生クラスはその基本クラスの public および protected の全てのメンバを継承します．つまり，派生クラスから見ると，基本クラス (スーパークラス) の public および protected のメンバは，あたかも派生クラスで宣言されたようになります．例えば，クラスXと派生クラスYが

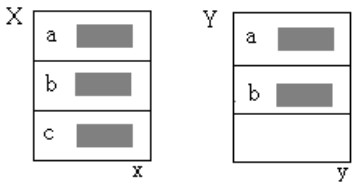
```
class X
{
        public int a;
        protected int b;
        private int c;
}

class Y extends X
{
        public int d;
}
```

で定義され，オブジェクト x と y が

```
        X x = new X();
```

Y y = new Y();

と宣言されたとします．これを図で表すと次のようになります．



クラスXの public メンバ a はYの public メンバとして継承され，クラスXの protected メンバ b は Y の protected メンバとして継承されますが，クラス X の private メンバ c はクラス Y に継承されません．

継承されたメンバのオーバーライド (Overriding Inherited Members)

X が基本クラス (スーパークラス) で，YがXのサブクラス (派生クラス) のとき，Yのオブジェクトは Xの全ての public または protected メンバ変数およびメンバ関数を継承します．例えば，上の例題の Person クラスのフィールド変数 name などです．

ところが，ときには継承されたメンバを派生クラスに合わせた形に定義したいときがあります．例えば，name が Person クラスのフィールド変数で，Student が Person クラスの派生クラスのとき，name というメンバ変数を Student クラスに新たに定義することができます．このとき，**Student で定義されたフィールド変数 name は Person クラスのフィールド変数 name に対し優位 (dominate) であるといいます．** ここで，クラス Student のオブジェクト s による name の参照 s.name は Person クラスで定義された name ではなく，Student で定義されたフィールド変数 name をアクセスします．Person クラスのフィールド変数 name をアクセスするときには **super** キーワードを用いて，super.name とします．つまり，Student クラスオブジェクトは基本クラス (スーパークラス) とサブクラスの両方のメンバにアクセスできます．

例題 7.3 フィールドのオーバーライド

Person クラスから Student クラスを派生させ，Person クラスのオブジェクト p と Student クラスのオブジェクト s を用意し，両方のクラスのメンバ変数 name にアクセスするプログラムを作成しなさい．

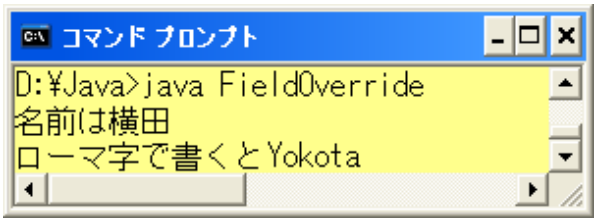
解答 Student クラスオブジェクト s が Student クラスのメンバ変数 name にアクセスするには，s.name でよい．また，Person クラスのメンバ変数にアクセスするには，super.name のように，super を用いる．

```
=====================================================
class Person
{
```

```
        protected String name, nationality;
        public Person(String name, String nationality)
        {
            this.name = name;
            this.nationality = nationality;
        }
}
class Student extends Person
{
        protected String id;
        protected double gpa;
        protected String name;
        protected String nameP;
        public Student(String name, String id, double gpa)
        {
            super("横田","日本");
            nameP = super.name;
            this.name = name;
            this.id = id;
            this.gpa = gpa;
        }
}
public class FieldOverride
{
        public static void main(String[] args)
        {
            Student st = new Student("Yokota","600813902",3.85);
            System.out.println("名前は" + st.nameP);
            System.out.println("ローマ字で書くと" + st.name);
        }
}
```

==========

実行結果



メソッドに対しても同じことがいえます．printName() メソッドが Person クラスで定義され，さらに Student クラスでも定義されたならば，Student クラスのオブジェクト s による呼び出し s.printName() は Student で定義された printName() メソッドを呼び出します．Person

クラスで定義された printName() メソッドを呼び出すには，キーワード **super** を用いて，super.printName() とします．このとき， メソッド s.printName() は printName() メソッドを**オーバーライド (override)** するといいます．

練習問題 7.1 printName() メソッドを含む Person クラスとその派生クラス Student クラスを用い，Student クラスオブジェクト s を用意し，s.printName() と super.printName() を比較しなさい．

派生クラスから基本クラスのメンバへのアクセス

すでに説明しましたように，private と protected の違いは，派生クラスから protected メンバはアクセスできますが private メンバはできないところです．では，いつメンバを private にしたらよいのでしょうか．その答えは，principle of information hiding(情報隠蔽原則) で説明されます．つまり，最初はアクセスを制限し，後で変更します．もし，将来，メンバ変数の変更を考えているのなら，private で宣言しておくことにより，派生クラスでの変更を必要のないものにします．派生クラスは private データメンバと独立です．

例えば，Person クラスのオブジェクトである人物が大卒かどうか知る必要があるとします．このとき，フィールド変数 college を protected で追加することができますが，後で，もっと詳しい学歴の情報を付け加えるかもしれないので，この段階では private メンバ変数にし，派生クラスからのアクセスを制限するほうがよいでしょう．

確認問題 7.1

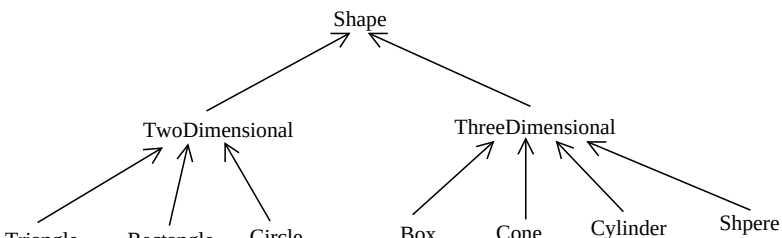
1. 合成と継承の違いは何か
2. proteced と private メンバの違いは何か
3. 仮想関数とは何か
4. 抽象基本クラスとは何か
5. 多相とは何か
6. 具象導出クラスとは何か
7. 静的結合と動的結合の違いは何か

演習問題 7.1

1. 次の定義はどこが間違っているか

```
class X
{
   protected:
      int a;
};
class Y : public X
{     public:
         void set(X x, int c) {x.a = c;}
};
```

2. 次のクラス階層を実装せよ.



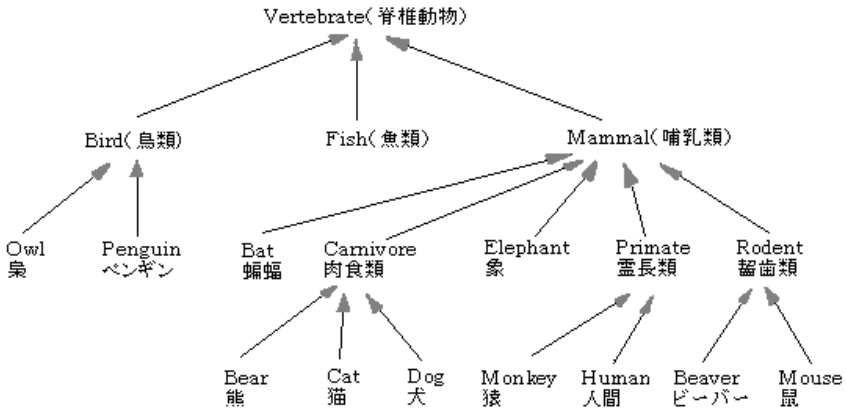
第 8 章

インターフェイス (Interface)

抽象クラス (Abstract Classes)

クラス階層の中でそれ自身に属するオブジェクトが存在せず，他のクラスの基本クラスとしてのみ定義されているクラスのことを抽象クラス (abstract class) といいます．

よいオブジェクト指向プログラムはクラスの階層を含み，階層関係は下の樹形図のような形で表されます．



この樹形図の葉の部分のクラス（例えば，Fish, Owl, Dog）は，それらのクラスの行動を実装するための特定なメソッドを含んでいます（例えば，Fish.swim( ), Owl.fly( ), Dog.run( )）．ところが，メソッドには全ての派生クラスでも共通なものもあります（例えば，Vertebrate.eat( ), Mammal.suckle( ), Primate.peel( )）．これらのメソッドは基本クラスで抽象メソッドとして宣言します．抽象メソッドにするには，**abstract** キーワードを用います．

クラス階層のそれぞれのクラスは，1 つでも抽象メソッドを含んでいれば，抽象クラスに，それ以外の場合は，具象クラスに分けられます．抽象クラス (abstract class) は 1 つ以上の抽象メソッドを含んでいるクラスで，具象派生クラス (concrete derived class) は 1 つも抽象メソッドを含まないクラスです．上の例では，Vertebrate クラスは抽象クラスで Fish クラスは具象派生ク

ラスです．抽象クラスのオブジェクトは宣言できません．

抽象クラスは他のクラスの基本クラスとなって，その派生クラスのインターフェイスや基本的なデータ構造を定義するためにだけ存在し，抽象クラス自身の型のオブジェクトを宣言することは誤りです．

ある派生クラスの基本クラスが抽象クラスである場合，その派生クラスの定義の中で基本クラスの抽象メソッドを全て再定義する必要があります．上の例で，Vertebrate.eat( ), Mammal.suckle( ), と Primate.peel( ) の 3 つのメソッドだけが抽象メソッドだとすると，抽象クラス (ABC) は Vertebrate, Mammal, と Primate で，残りの１５クラスは具象派生クラス (CDC) となります．そして，これらの１５の派生クラスでは，それ自身の eat( ) 関数の実装をし，Mammal の１１個の具象派生クラスはそれ自身の suckle( ) 関数の実装をし，２つの Primate クラスの派生クラスでは peel( ) 関数の実装をする必要があります．

抽象基本クラスは，概してクラス階層の構築の最初の段階で定義されます．そして，抽象基本クラスをもとに，派生クラスの骨組みができあがります．そのとき，抽象メソッドは階層におけるある種の均一性を規定します．

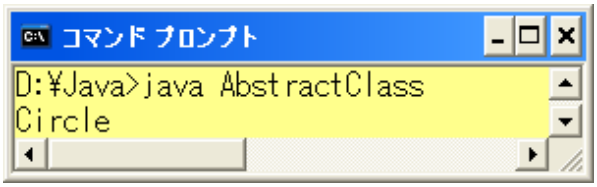
例題 8.1 抽象クラス
抽象メソッド display() を宣言する抽象クラス Shape を作成し，Circle クラスによって拡張しなさい．

```
=====================================================
abstract class Shape
{
        abstract void display();
}
class Point3D
{
        double x,y,z;
        public Point3D(double x, double y, double z)
        {
            this.x = x;
            this.y = y;
            this.z = z;
        }
}
class Circle extends Shape
{
        Point3D center, p;
        public Circle(Point3D center, Point3D p)
        {
            this.center = center;
            this.p = p;
```

```
        }
        public void display()
        {
            System.out.println("Circle");
        }
}
class AbstractClass
{
        public static void main(String[] args)
        {
            Circle c = new Circle(new Point3D(0,0,0), new Point3D(1,0,0));
            c.display();
        }
}
```

==========

実行結果



インターフェイス (Interface)

Java は言語構造の簡潔さを指向しているため，同時に二つ以上の基本クラスを持つ多重継承を記述することができません．このため，抽象メソッドは特定の基本クラスの内部に含ませる形でしか利用できません．また，継承を重ねた間接継承で対応することもできますが，それはプログラムを分かり難くさせます．これを解決するためいに Java は，**インターフェイス**という機能を用意しました．インターフェイスを用いると，多重継承に近い処理が可能となります．

インターフェイスは，クラスメンバを操作する手順としてのメソッドと static 変数がたくさん集まったものです．インターフェイスの記述は，abstract と class の代わりに **interface** キーワードを用いて行います．

```
interface 名前
{
        型　フィールド名　= 初期値;
        型　メソッド名 (引数);
}
```

この形式には abstract キーワードがありませんが，インターフェイス内では，フィールドは public static final，メソッドは abstract public で解釈されることになっています．インターフェ

イスを用いるときに注意するとことは，以下の通りです．

- インターフェイスにはいかなる実装も含まれない
- インターフェイス内のメソッドは暗黙の内に abstract であり，public である
- インターフェイスにクラスメソッドを含むことはできない
- インターフェイス内のフィールドは定数だけ定義できる
- インターフェイスはインスタンスを生成できない
- 一般の基本クラスと同じように，継承したサブクラスを，インターフェイス型の変数で操作することはできる

例題 8.2 インターフェイス
Shape2D インターフェイスでは，閉じられた 2 次元の図形の面積を計算して返す getArea() メソッドを宣言し，Shape3D インターフェイスでは，閉じられた 3 次元の図形の体積を計算して返す getVolume() メソッドを宣言しなさい．

```
=====================================================
interface Shape2D
{
        double getArea();
}
interface Shape3D
{
        double getVolume();
}

=====================================================
```

インターフェイスの実装

インターフェイスは単独でクラスに実装することもできますし，複数のインターフェイスを実装することもできます．また，クラスの継承とインターフェイスの実装を同時に指定できます．この場合も，直属の基本クラスはただ一つしか持つことができません．

例題 8.3 インターフェイスの実装
Circle クラスに Shape2D インターフェイスを実装し，Shapes クラスでこれらのクラスをインスタンス化し，そのメソッドの動作をテストしなさい．

```
=====================================================
interface Shape2D
{
        double getArea();
}
```

```
abstract class Shape
{
        abstract void display();
}
class Point3D
{
        double x,y,z;
        public Point3D(double x, double y, double z)
        {
            this.x = x;
            this.y = y;
            this.z = z;
        }
}
class Circle extends Shape implements Shape2D
{
        Point3D center, p;
        public Circle(Point3D center, Point3D p)
        {
            this.center = center;
            this.p = p;
        }
        public void display()
        {
            System.out.println("Circle");
        }
        public double getArea()
        {
            double dx = center.x - p.x;
            double dy = center.y - p.y;
            double d = dx*dx + dy*dy;
            double radius = Math.sqrt(d);
            return Math.PI*radius*radius;
        }
}
class Shapes
{
        public static void main(String[] args)
        {
            Circle c = new Circle(new Point3D(0,0,0), new Point3D(1,0,0));
            c.display();
            System.out.println("面積は" + c.getArea());
        }
}
```

==========

実行結果

インターフェイスの参照

インターフェイスの名前を変数の型として指定することができます．この変数は，そのインターフェイスを実装したオブジェクトを参照するために用いることが可能です．そのインターフェイスを実装していないオブジェクトを変数に代入すると，コンパイルエラーになります．

例題 8.4 インターフェイスの参照

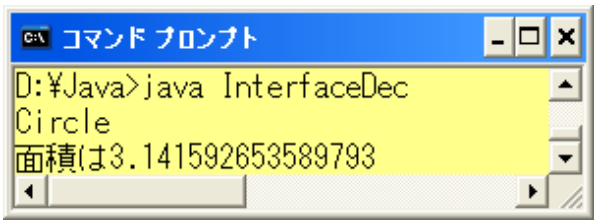
Circle クラスに Shape2D インターフェイスを実装し，InterfaceDe クラスで Shape インターフェイス変数 sh を宣言し，Circle クラスオブジェクトを代入し，面積を計算するメソッドの動作をテストしなさい．

```
=====================================================
interface Shape2D
{
        double getArea();
}
abstract class Shape
{
        abstract void display();
}
class Point3D
{
        double x,y,z;
        public Point3D(double x, double y, double z)
        {
            this.x = x;
            this.y = y;
            this.z = z;
        }
}
class Circle extends Shape implements Shape2D
{
        Point3D center, p;
        public Circle(Point3D center, Point3D p)
        {
```

```
        this.center = center;
        this.p = p;
    }
    public void display()
    {
        System.out.println("Circle");
    }
    public double getArea()
    {
        double dx = center.x - p.x;
        double dy = center.y - p.y;
        double d = dx*dx + dy*dy;
        double radius = Math.sqrt(d);
        return Math.PI*radius*radius;
    }
}
class InterfaceDec
{
    public static void main(String[] args)
    {
        Circle c = new Circle(new Point3D(0,0,0), new Point3D(1,0,0));
        c.display();
        Shape2D sh;
        sh = c;
        System.out.println("面積は" + sh.getArea());
    }
}
```

==========

実行結果



instanceof 演算子

プログラムの都合上，オブジェクトのクラスや，オブジェクトに実装されているインターフェイスを確認しなければならないことがあります．例えば，

```
class SubClass implements Intf...
```

という定義があるとき，

```
SubClass dt = new SubClass();
```

if(dt **instanceof** Intf)

と書くと，if 文の条件式は真になります．

パッケージ

Java ではコンパイル時にソースファイル上のクラスファイルが作成されます．したがって，異なるプログラムでも同じクラス名を用いていると，クラスファイル名が衝突して，期待した動作が得られなくなります．この問題に対処するために Java には，**パッケージ (package)** というファイル管理機能があります．パッケージを用いると関連のあるファイルを隔離して管理するので，名前の重複に対応できるようになります．つまり，自分で新たなディレクトリ（フォルダ）を作成し，プログラムごとに格納する代わりに，Java では言語機能として，新たなディレクトリを作成し，そこにクラスファイルを格納してくれるのです．

パッケージの指定方法は，ソースファイルの先頭に

package パッケージ名;

と書きます．コンパイルの方法は，

javac パッケージ名

ファイル名.java

となります．実行方法は，

java パッケージ名. ファイル名となります．

複数のソースファイルを同じパッケージに入れる

複数のソースファイルを同じパッケージに入れるには，

- それぞれのファイルの先頭に同じパッケージ指定を入れる
- それぞれのファイルを指定されたパッケージに保存する

複数のソースファイルを別々のパッケージに入れる

複数のソースファイルを別々のパッケージに入れるには，

- それぞれのファイルの先頭にパッケージ指定を入れる
- 他のパッケージから呼び出されるクラスには public 属性をつける
- 他のパッケージにあるクラスを呼び出すときには「パッケージ名.」をつける

import ステートメント

同じパッケージ内のクラスやインターフェイスにアクセスするのは簡単です．その名前を直接指定するだけでアクセルできます．しかし，別のパッケージ内の型にアクセスする場合は，名前を直接指定できません．この問題を解決する 1 つの方法は，**完全修飾名**を使って型を指定することです．例えば，java.awt.event は Java の組み込みパッケージの 1 つです．このパッケージには，MouseEvent クラスが定義されています．したがって，java.awt.event 内の MouseEvent クラスにアクセスするには，java.awt.event.MouseEvent と記述します．

しかし，完全修飾名を頻繁に使うのは面倒です．そこで，Java には「インポート」と呼ばれる機能が用意されています．インポート機能を用いるとパッケージ名を個別付与する必要がなくなります．インポート宣言は次のように行います．

import パッケージ名．クラス名；

これにより，コンパイラにクラスが格納されている場所を知らせます．

第 9 章

例外 (Exception)

例外とは，プログラム中に発生した問題を通知するために，実行時に生成されるオブジェクトのことです．例外が発生する典型的なケースは，整数をゼロで乗算した，配列のインデックスが負の値または配列の長さを超える値だった，ファイルが見つからなかった，数値の形式が不正だった，URL が壊れていた，などの場合です．

この種の問題の処理は，Java 言語の重要な機能です．例外を適切に処理することで，プログラムの構造はより堅牢になります．

例外処理

Java 言語では，プログラムの実行中に発生した例外を，ユーザの用意した処理ルーチンで処理することができます．それには，try/catch/finally 文を用います．

```
try {
    // 例外を検出する文をここに記入
}
catch {
    // 例外が発生したときの処理をここに記入
}
finally {
    // 例外発生の有無に関わらず必ず実行する文をここに記入
}
```

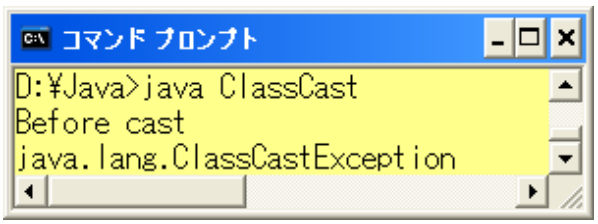
例題 9.1 例外処理
次のプログラムを実行すると，どのような情報が表示されるか答えなさい．

```
==================================================
class ClassCast

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

```
try

    Object obj = new Integer(”85”);
    System.out.println(”Before cast”);
    Double dobj = (Double)obj;
    System.out.println(”After cast”);

catch(Exception ex)

    System.out.println(ex);
```

==========

実行結果



ここで，用いた catch(Exception ex) は，とりあえずどのような例外が発生しているか分からないが，全ての例外を捕捉したいときに用います．

チェック例外の処理

例外には，チェック例外と非チェック例外の 2 つがあります．チェック例外とは，正しく記述されているかどうかをコンパイラがチェックするものです．非チェック例外は例外の有無をコンパイラがチェックすることはないものです．なぜ，このような違いがあるかといいますと，チェック例外とは，コンストラクタ/メソッドを実行すると発生する可能性がある例外で，非チェック例外とは，Java の通常の式の実行で発生する可能性がある例外です．

非チェック例外の代表例としてゼロ除算 (Arithmetic Exception) があります．これは，

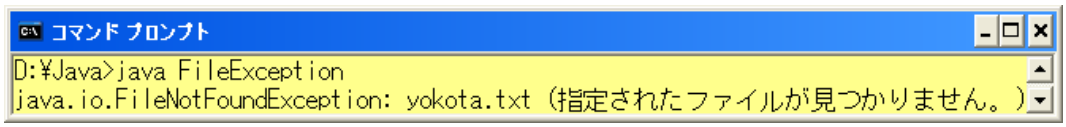
$$a = b/c;$$

という，ごくありふれた式文の中で発生する可能性があります．Java ではこのような例外はコンパイラには判断させず，プログラマ自身が対処すべきであるとしています．

例題 9.2 メソッド内での例外処理
次のプログラムを実行すると，どのような情報が表示されるか答えなさい．

```
=====================================================
import java.io.*;
class ThrowsException
{
        public static void main(String[] args)
        {
            fileopen();
        }
        public static void fileopen()
        {
            try
            {
                FileReader f = new FileReader(”yokota.txt”);
            }
            catch(Exception ex)
            {
                System.out.println(ex);
            }
        }
}
=====================================================
```

実行結果



メソッド外での例外処理

もし例外を検出したメソッド内で処理をしないときは，自分を呼び出した側に例外処理を依頼します．その場合は **throws** キーワードを用いてスロー宣言をします．これは，このメソッドから例外が発生することがある．したがって呼び出し側では責任を持って例外処理して欲しいということを表明するものです．

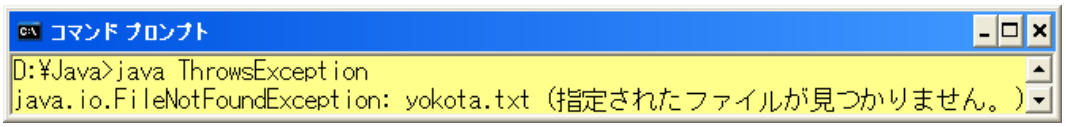
例題 9.3 メソッド外での例外処理
次のプログラムを実行すると，どのような情報が表示されるか答えなさい．

```
=====================================================
import java.io.*;
class ThrowsException
{
        public static void main(String[] args)
```

```
        {
            try
            {
                fileopen();
            }
            catch(Exception ex)
            {
                System.out.println(ex);
            }

          }
        public static void fileopen() throws Exception
        {
            FileReader f = new FileReader("yokota.txt");
        }
}
```

==========

実行結果



第 10 章

Java クラスライブラリ

熟練 Java プログラマになるには，Java 言語とクラスライブラリの両方をよく理解する必要があります．Java には 8 個の基本データ型のそれぞれに対応する 8 個のラッパークラス (Boolean, Character, Byte, Short, Integer, Long, Float, Double) が定義されています．これらは，それぞれ，boolean, char, byte, short, int, long, float, double の基本データ型の各値をカプセル化するものです．

ラッパークラスの主要な利点は，それらが文字列を基本データ型に変換するメソッドを提供していることです．つまり，データは普段文字列として扱い，必要に応じて変えることができるということです．そこで，まず Java の文字列の処理について学びます．

String クラス

String クラスを用いると，可変長の文字列を操作できますので，ユーザは格納される文字列の長さに注意する必要はありません．

String クラスの主なコンストラクタには以下のものがあります．

String()	空の文字列を作成する
String(byte[] buffer)	byte 配列から文字列を作成する
String(String str)	文字列 str の内容で新しい文字列を作成

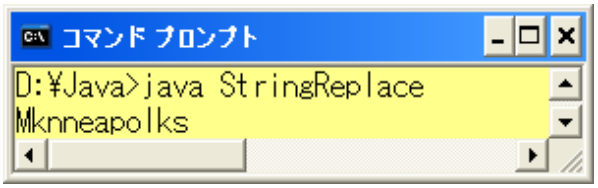
String クラスの主なインスタンスメソッドには以下のものがあります．

int indexOf(String str)	文字列 str が最初に現われる位置を返す
int lastIndexOf(String str)	文字列 str が最後に現われる位置を返す
String replace(char oldChar, char new Char)	文字列内の文字 oldChar を newChar で置き換え
String substring(int beginIndex)	位置 beginIndex 以降の文字を返す
String toLowerCase()	文字列を小文字に変換する
String toUpperCase()	文字列を大文字に変換する

例題 10.1 Random クラス
文字列 Minneapolis の i を k に置き換えるプログラムを作成しなさい.

```
==========================================================
class StringReplace
{
        public static void main(String[] args)
        {
            String str = "Minneapolis";
            str = str.replace('i','k');
            System.out.println(str);
        }
}

==========================================================
```

実行結果



StringBuffer クラス

StringBuffer クラスは文字列自身を変更する必要のある処理に用いられます.

String クラスの主なコンストラクタには以下のものがあります.

StringBuffer()	16 文字の初期容量を持つ空の文字列バッファを作成する
StringBuffer(int len)	len で示す文字数の初期容量を持つ空の文字列バッファを作成する
StringBuffer(String str)	文字列 str から文字列バッファを作成

String クラスの主なインスタンスメソッドには以下のものがあります.

StringBuffer append(String str)	文字列 str を追加する
StringBuffer delete(int start, int end)	位置 start から end-1 までの文字列を削除
StringBuffer insert(int offser, String str)	位置 offset に文字列 str を挿入
StringBuffer length()	文字列バッファの長さを返す
StringBuffer substring(int start)	位置 start から末尾までの文字列を返す
String toString()	文字列バッファのデータを現す文字列を返す

char 配列，byte 配列，StringBuffer とのデータ交換 String オブジェクトと，char[] 型，byte[] 型，あるいは Stringbuffer 型との相互データ交換を行うことができます．このとき基本データ型の配列は固定長なので，その長さに注意が必要です.

char[],byte[] から String へのデータ交換

```
char[] ch = 'a','b';
byte[] bt = 'A','B';
String s1 = new String(ch);//char[] から String へ
String s2 = new String(bt);//byte[] から String へ
```

String と StringBuffer のデータ交換

```
StringBuffer sb = new StringBuffer();
String s = sb.toString();//StringBuffer から String へ
sb = new StringBuffer(s);//String から StringBuffer へ
```

Integer 系ラッパークラス

Integer は Java のクラスライブラリの中でも特によく用いられるクラスの 1 つです.

Integer クラスには，MAX_VALUE と MIN_VALUE の 2 つのクラス変数が定義されています.

Integer クラスの主なクラスメソッドには以下のものがあります.

static Integer decode(String s)	s を基数 10 の値として変換して Integer オブジェクトを返す
static int parseInt(String s)	s を基数 10 の値として変換して int 型を返す
static int parseInt(String s, int r)	s を基数 r の値として変換して int 型を返す

Integer クラスの主なインスタンスメソッドには以下のものがあります.

byte byteValue()	現在のオブジェクトの値を byte 型で返す
double doubleValue()	現在のオブジェクトの値を double 型で返す
boolean equals(Object obj)	obj と現在のオブジェクトが同じ値なら真を返す
int intValue()	現在のオブジェクトの値を int 型で返す
long longValue()	現在のオブジェクトの値を long 型で返す
short shortValue()	現在のオブジェクトの値を short 型で返す
String toString()	現在のオブジェクトの値の文字列表現を返す

例題 10.2 Integer クラス

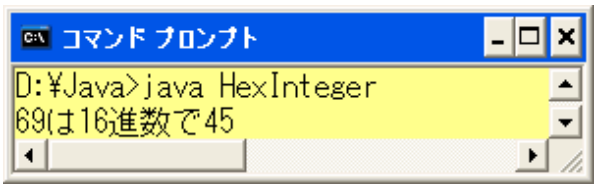
16 進数で 45 を表す数を表示するプログラムを作成しなさい.

```
=====================================================
class HexInteger
{
    public static void main(String[] args)
    {
        String str = "45";
        System.out.println(Integer.parseInt(str,16) + "は 16 進数で" + str);
```

```
        }
}
```

==========

実行結果



Double 系ラッパークラス

浮動小数点を扱うものに，Float ラッパークラスと Double ラッパークラスがあります．この 2 つは内容がよく似ているので，ここでは Double ラッパークラスについて学びます．

Double クラスには，MAX_VALUE，MIN_VALUE，NaN，NEGATIVE_INFINITY，POSITIVE_INFINITY，TYPE の 6 つのクラス変数が定義されています．

Double クラスの主なクラスメソッドには以下のものがあります．

boolean isInfinite(double d)	d の値が無限なら真を返す
boolean isNaN(double d)	d の値が非数値なら真を返す
String toString(double d)	d の値の文字列表現を返す
Double ValueOf(String s)	s の表す float 値をカプセル化した Double オブジェクトを返す

Double クラスの主なインスタンスメソッドには以下のものがあります．

byte byteValue()	現在のオブジェクトの値を byte 型で返す
double doubleValue()	現在のオブジェクトの値を double 型で返す
float floatValue()	現在のオブジェクトの値を float 型で返す
int intValue()	現在のオブジェクトの値を int 型で返す
boolena isInfinite()	値が NEGATIVE_INFINITY または POSITIVE_INFINITY なら真
short shortValue()	現在のオブジェクトの値を short 型で返す
String toString()	現在のオブジェクトの値の文字列表現を返す

例題 10.3 Double クラス

3 を 0 で割る場合，クラスメソッドの isInfinity() と isNaN() を用いて結果をテストするプログラムを作成しなさい．

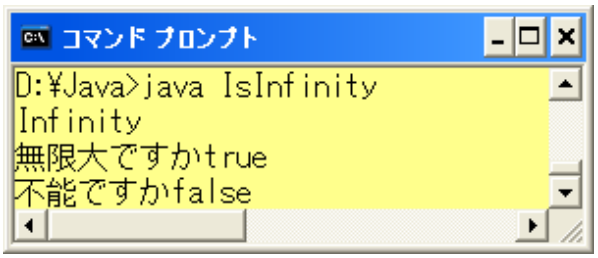
==========

class IsInfinity

```
{
        public static void main(String[] args)
        {
            double divident = 3;
            double divisor = 0;
            double q = divident/divisor;
            System.out.println(q);
            // 無限と NaN のテスト
            System.out.println("無限大ですか"+ Double.isInfinite(q));
            System.out.println("不能ですか"+ Double.isNaN(q));
        }
}
```

==========

実行結果



Character 型ラッパークラス

Characte クラスは char 型の値をカプセル化するものです.

Character クラスの主なクラスメソッドには以下のものがあります.

boolean isDigit(char c)	c が数字なら真をそうでなければ偽を返す
boolean isLetter(char c)	c が文字なら真をそうでなければ偽を返す
boolean isLetterOrDigit(char c)	c が文字または数字なら真をそうでなければ偽を返す
boolean isLowerCase(char c)	c が小文字なら真をそうでなければ偽を返す
boolean isSpaceChar(char c)	c がスペース文字なら真をそうでなければ偽を返す
boolean isWhiteSpace(char c)	c がホワイトスペースなら真をそうでなければ偽を返す
char toLowerCase(char c)	c を小文字に変換して返す

Character クラスの主なインスタンスメソッドには以下のものがあります.

char charValue()	現在のオブジェクトの値を char 型で返す
boolean equals(Object obj)	obj と現在のオブジェクトが同じ値なら真を返す
int digit()	現在のオブジェクトの値を int 型で返す

Byte 型ラッパークラス

Byte クラスは byte 型の値をカプセル化するものです.

Byte クラスの主なクラスメソッドには以下のものがあります.

Byte decode(String s)	s の表す値をカプセル化した Byte オブジェクトを返す
byte parseByte(String s)	s を基数 10 の値として変換して byte 型を返す
byte parseByte(String s, int r)	s を基数 r の値として変換して byte 型を返す

Byte クラスの主なインスタンスメソッドには以下のものがあります.

byte byteValue()	現在のオブジェクトの値を byte 型で返す
double doubleValue()	現在のオブジェクトの値を double 型で返す
float floatValue()	現在のオブジェクトの値を float 型で返す
int intValue()	現在のオブジェクトの値を int 型で返す
long longValue()	現在のオブジェクトの値を long 型で返す
short shortValue()	現在のオブジェクトの値を short 型で返す
String toString()	現在のオブジェクトの値の文字列表現を返す

Short 型ラッパークラス

Short クラスは short 型の値をカプセル化するものです.

Short クラスの主なクラスメソッドには以下のものがあります.

Short decode(String s)	s の表す値をカプセル化した Short オブジェクトを返す
short parseShort(String s)	s を基数 10 の値として変換して short 型を返す
short parseShort(String s, int r)	s を基数 r の値として変換して short 型を返す

Byte クラスの主なインスタンスメソッドには以下のものがあります.

byte byteValue()	現在のオブジェクトの値を byte 型で返す
double doubleValue()	現在のオブジェクトの値を double 型で返す
float floatValue()	現在のオブジェクトの値を float 型で返す
int intValue()	現在のオブジェクトの値を int 型で返す
long longValue()	現在のオブジェクトの値を long 型で返す
short shortValue()	現在のオブジェクトの値を short 型で返す
String toString()	現在のオブジェクトの値の文字列表現を返す

Long 型ラッパークラス

Long クラスは long 型の値をカプセル化するものです.

Long クラスの主なクラスメソッドには以下のものがあります.

long parseLong(String s)	s を基数 10 の値として変換して long 型を返す
long parseLong(String s, int r)	s を基数 r の値として変換して long 型を返す
String toBinaryString(long l)	l の値の 2 進数表現を返す
String toHexString(long l)	l の値の 16 進数表現を返す
String toOctalString(long l)	l の値の 8 進数表現を返す
String toString(long l)	l の値の文字列表現を返す

Long クラスの主なインスタンスメソッドには以下のものがあります．

byte byteValue()	現在のオブジェクトの値を byte 型で返す
double doubleValue()	現在のオブジェクトの値を double 型で返す
float floatValue()	現在のオブジェクトの値を float 型で返す
int intValue()	現在のオブジェクトの値を int 型で返す
long longValue()	現在のオブジェクトの値を long 型で返す
short shortValue()	現在のオブジェクトの値を short 型で返す
String toString()	現在のオブジェクトの値の文字列表現を返す

System クラス

System クラスには，ランタイム環境に関するいくつかの属性が定義されています．1 つは，すでに使用している **out** というクラス変数です．この変数には **PrintStream** オブジェクトへの参照が入っていて，そのオブジェクトの **print()** と **println()** メソッドで標準出力に文字列引数を表示することになります．クラス変数 **err** にも PrintStream オブジェクトへの参照が入っています．これは標準エラーストリームです．クラス変数 **in** には **InputStream** オブジェクトへの参照が入っています．PrintStream と InputStream は入出力をサポートするクラスです．

System くらすのもう 1 つのクラスメソッドに **exit()** があります．これは現在のプログラムを終了します．

System クラスの主なクラスメソッドには以下のものがあります．

long currentTimeMillis()	現在の時刻を返す
void exit(int status)	status を終了コードとして JVM を終了
void gc()	ガベージコレクタを実行
void load(String filename)	filename で指定したダイナミックライブラリを読み込む
String toOctalString(long l)	l の値の 8 進数表現を返す
String toString(long l)	l の値の文字列表現を返す

Math クラス

Math クラスは数学関係のメソッドを集約させたものです．すべて static メソッドなので，「Math.sin()」のように簡単に使用できます．Math クラスには円周率や自然対数の底 e といった定数も定義されています．System くらすのもう 1 つのクラスメソッドに **exit()** があります．こ

れは現在のプログラムを終了します．

System クラスの主なクラスメソッドには以下のものがあります．double の場合だけをのせていますが，float,int,long でも使える多重定義となっています．

double abs(double a)	double 値の絶対値を返す
double acos(double a)	アークコサイン値を返す
double asin(double a)	アークサイン値を返す
double atan(double a)	アークタンジェント値を返す
double ciel(double a)	引数の小数点以下を切り上げた値を返す
double floor(double a)	引数の小数点以下を切り捨てた値を返す
double cos(double a)	コサイン値を返す
double sin(double a)	サイン値を返す
double tan(double a)	タンジェント値を返す
double min(double a, double b)	小さい方の値を返す
double max(double a, double b)	大きい方の値を返す
double pow(double a, double b)	a を b 乗した値を返す
double exp(double a)	e^a の値を返す
double log(double a)	$\log a$ の値を返す

次に，Java.util パッケージに含まれる，乱数，日付，ベクトル，ハッシュ表，スタックなどについて学びます．

Random クラス

Random クラスは，double 型，float 型，int 型，または long 型の乱数を生成するために用います．また，ガウス分布の乱数を生成することもできます．この機能は，現実の世界のシステムをシミュレートする場合に役に立ちます．

Random クラスには，次のコンストラクタがあります．

```
Random();
Random(long seed);
```

Random クラスの主なインスタンスメソッドには以下のものがあります．

void nextBytes(byte[] buffer)	buffer を乱数で埋める
double nextDouble()	double 型の乱数を返す
float nextFloat()	float 型の乱数を返す
double nextGaussian()	標準正規分布の乱数を返す
int nextInt()	int 型の乱数を返す
logn nextLong()	long 型の乱数を返す
void setSedd(long seed)	乱数ジェネレータに種を与える

例題 10.4 Random クラス
10 個の int 型の乱数を生成するプログラムを作成しなさい.

```
==========================================================
import java.util.*;
class RandomInt
{
        public static void main(String[] args)
        {
            Random r = new Random();
            for(int i=0; i ¡ 10; i++)
            {
                System.out.println(r.nextInt());
            }
        }
}

==========================================================
```

実行結果

Date クラス

Date クラスは，特定の日付と時刻に関する情報を得るときに用います.

Date クラスには，次のコンストラクタがあります.

Date();
Date(long msec);

1 つ目のコンストラクタでは，現在の日付と時刻を表すオブジェクトが返されます．2 つ目のコ

ンストラクタでは，基準時 (1970 年 1 月 1 日) からの経過時間をミリ秒で表すオブジェクトがかえされます．

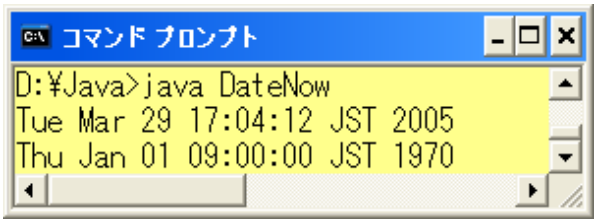
Date クラスの主なインスタンスメソッドには以下のものがあります．

boolean after(Date d)	d が現在の日付より後ならば真を返す
boolean before(Date d)	d が現在の日付より前ならば真を返す
boolean equal(Date d)	d が現在の日付と同じならば真を返す
double nextGaussian()	標準正規分布の乱数を返す
long getTime()	基準時からの経過時間をミリ秒で返す
void setTime(long msec)	基準時から msec 経過した日付と時刻をオブジェクトに設定
String toString()	日付を文字列にして返す

例題 10.5 Random クラス
現在の日付と時刻を表示するプログラムを作成しなさい．

```
==========================================================
import java.util.*;
class DateNow
{
        public static void main(String[] args)
        {
            Date currentDate = new Date();
            System.out.println(currentDate);
            Date baseDate = new Date(0);
            System.out.println(baseDate);
        }
}

==========================================================
```

実行結果



第 11 章

入出力

ここでは，**java.io** パッケージで最もよく利用されるクラスについて学びます．まず，最初にファイルとディレクトリを扱う方法を学びます．次に，**ストリーム**の作成および使用方法について学びます．

ファイル

Java にはファイルの入出力を行うのではなく，ファイルそのものを管理する **File** クラスが用意されています．File クラスは，ファイルまたはディレクトリの**プロパティ**に関する情報をカプセル化します．

File クラスのコンストラクタ

File(String path)

File(String directoryPath, String filename)

File(File directory, String filename)

File クラスには指定したファイルの状態を参照する多くのメソッドが用意されています．これらのメソッドは例外スローを行わないので，記述も簡単です．

状態参照メソッド	
boolean canRead()	ファイルが存在し読み取り可能な場合真を返す
boolean canWrite()	ファイルが存在し書き込み可能な場合真を返す
boolean exists()	ファイルが存在する場合真を返す
String getAbsolutePath()	ファイルの絶対パスを返す
String getCanonicalPath()	ファイルの標準パスを返す
String getName()	ファイルの名前を返す
String getParent()	ファイルの親を返す
String getPath()	ファイルのパスを返す
boolean isDirectory()	ファイルがディレクトリの場合真を返す
boolean isFile()	ファイルがディレクトリでない場合真を返す
long length()	ファイルのサイズをバイトで返す
long lastModified()	基準時から前回の変更時刻までの時間をミリ秒で返す
String[] list()	ディレクトリに入っているファイルの名前を返す

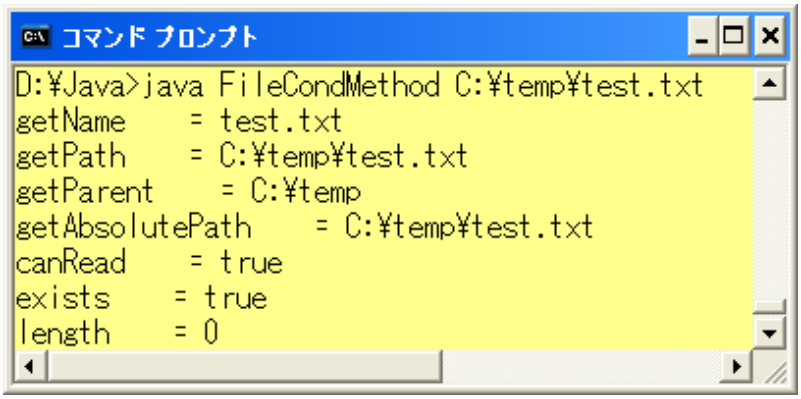
例題 11.1 File の管理

C:¥temp に test.txt ファイル作成し，そのファイルの状態を表すメソッドを用いて，ファイル名，パス名，パス名の中の親ディレクトリ，絶対パス，読み込み可能，ファイルが存在するか，ファイルのサイズを表示するプログラムを作成しなさい．ただし，ファイル名はコマンドライン引数として入力する．

```
==================================================
import java.io.*;
class FileCondMethod
{
        public static void main(String[] args)
        {
            if(args.length == 0) System.exit(1);
            //ファイルを設定
            File mf = new File(args[0]);

            System.out.println("getName = " + mf.getName());
            System.out.println("getPath = " + mf.getPath());
            System.out.println("getParent = " + mf.getParent());
            System.out.println("getAbsolutePath = " + mf.getAbsolutePath());
            System.out.println("canRead = " + mf.canRead());
            System.out.println("exists = " + mf.exists());
            System.out.println("length = " + mf.length());
        }
}
```

==========

実行結果



File クラスにはいくつかの状態変更メソッドが用意されています．これらのメソッドは IOException 例外をスローするので，try 文で対応する必要があります．

状態変更メソッド	
boolean createNewFile()	正しくファイルが作成された場合真を返す
boolean delete()	ファイルが正しく削除されたら真を返す
boolean mkdir()	ディレクトリが作成されたら真を返す
boolean mkdirs()	ディレクトリが作成されたら真を返す
boolean renameTo(File newName)	ファイル名を変更が正しくできたら真を返す

例題 11.2 File クラス

C:¥temp に test.txt ファイル作成し，そのファイルの存在をテストし，新しいファイルを作成し，新規作成ができたか確認する．また，ファイル名を変更し，元のファイルが存在するか確認するプログラムを作成しなさい．

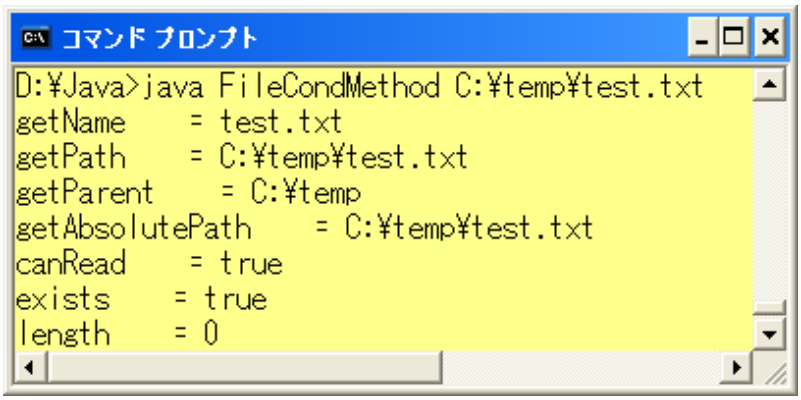
==========

```
import java.io.*;
class FileChangeMethod {
        public static void main(String[] args)
        {
            if(args.length == 0) System.exit(1);
            try
            {
            //ファイルを設定
                File mf = new File(args[0]);
```

```
            File mff = new File("mff.txt");

            System.out.println("exists(mf) = " + mf.exists());
            System.out.println("createNewFile = " + mf.createNewFile());
            System.out.println("exists(mf) = " + mf.exists());
            System.out.println("renameTo = " + mf.renameTo(mff));
            System.out.println("exists(mf) = " + mf.exists());
            System.out.println("exists(mff) = " + mff.exists());
        }
        catch(IOException ex)
        {
            System.out.println("err: " + ex);
        }
    }
}
```

==========

実行結果



ストリーム

プログラムでは，データの入力や出力という処理がよく用いられます．このデータの入出力を Java では**ストリーム (stream)** という概念で操作します．ストリームは入出力先を読み書き可能な状態に設定し，実際の読み書きを行い，その後閉じます．また，必要に応じて読み書き位置にマークしたり，書き込み中のデータを強制出力したりすることもできます．

実際にストリームを用いるときには，入力用または出力用ストリームのオブジェクトを宣言することで，論理デバイスとと物理デバイスを結び付けます．あとは，ユーザはストリームオブジェクトを使用することで，簡単にデータの入出力をすることができるようになります．

ストリームには，**文字ストリーム**と**バイトストリーム**の 2 つのタイプがあります．バイトストリームでは，**バイナリデータ**の読み取りおよび書き込むをおこなうことができます．文字スト

リームでは，文字および文字列の読み取りおよび書き込むを行うことができます．入力文字ストリームはバイトを文字に変換し，出力ストリームは文字をバイトに変換します．

Java では，内部で文字を 16 ビットの Unicode エンコーディングに従ってあらわします．ただし，このエンコーディングはマシンで使用されているエンコーディングと異なる場合があります．文字ストリームは，これらの 2 つのエンコーディング間で変換を行います．

文字ストリーム	
Reader 入力系の抽象基本クラス	Writer 出力系の抽象基本クラス
BufferedReader バッファ付文字入力	BufferedWrite バッファ付文字出力
InputStreamReader バイト入力を文字に変換 FileReader ファイルからの文字入力	OutputStreamWriter 文字をバイト出力に変換 FileWriter ファイルへの文字出力
	PrintWriter 型を持つデータの出力

バッファなし文字ストリーム

バッファなし文字ストリームは，FileWriter クラスと FileReader クラスを用いて行います．ファイル名を指定してインスタンスを生成するには，次の記述を行います．

書き込みファイル指定

FileWriter fw = new FileWriter(”test.txt”);

読み込みファイル指定

FileReader fr = new FileReader(”test.txt”);

これでファイルがオープンし，入出力の準備ができたことになります．このようにしてオープンしたファイルに対して，入出力処理を行います.. 処理が終了したら，fw.close(); と fr.close(); でファイルを閉じます．

データの読み込みを行うには，次の Reader クラスのインスタンスメソッドを用います．

Reader クラスのインスタンスメソッド	
void close() int read()	入力ストリームを閉じる ストリームから 1 文字を読み取る
int read(char[] buffer)	buffer に buffer.length 文字まで読み取ることを試みる．
int read(char[] buffer,int offset, int numChars)	buffer[offset] を先頭として， buffer に buffer.length 文字まで読み取る

データの書き込みを行うには，次の Writer クラスのインスタンスメソッドを用います．

Reader クラスのインスタンスメソッド	
void close()	出力ストリームを閉じる
void write(int c)	ストリームに c の下位 16 ビットを書き込む
void write(char[] buffer)	ストリームに buffer 内の文字を書き込む
void write(char[] buffer,int offset, int numChars)	buffer[offset] を先頭として， buffer に buffer.length 文字まで書き込む
void write(String s)	ストリームに s を書き込む
void write(String sm int index, int size)	ストリームに位置 index を先頭として s 内の size 文字を書き込む

例題 11.3 バッファなしファイル入出力
書き込みファイルを”test1.txt”とし，文字列”abcde”を書き込み，その後，書き込んだ文字列を読み取り表示し，ファイルを閉じるプログラムを作成しなさい．

```
=====================================================
import java.io.*;
class FileIO
{
        public static void main(String[] args)
        {
            String str = ”abcde”;
            try
            {
                //FileWriter オブジェクトの作成
                FileWriter fw = new FileWriter(”test1.txt”);
                //ファイルに文字列を書き込む
                fw.write(str);
                fw.write(”¥r¥n”);
                //FileWriter オブジェクトを閉じる
                fw.close();
            }
            catch(Exception ex)
            {
                System.out.println(”Exception: ” + ex);
            }
            try
            {
                //FileReader オブジェクトの作成
```

```
            FileReader fr = new FileReader(”test1.txt”);
            //文字を読み取って表示
            int i;
            while((i = fr.read()) != -1)
            {
                System.out.print((char)i);
            }
            //FileReader オブジェクトを閉じる
            fr.close();
        }
        catch(Exception ex)
        {
            System.out.println(”Exception: ”+ ex);
        }
    }
}
```

==========

実行結果

バッファあり文字ストリーム

バッファあり文字ストリームは，BufferedWriter クラスと BufferedReader クラスを用いて行います．バッファリングの利点は，物理デバイスへの読み取りおよび書き込みの回数を減らすことができるところです．ファイル名を指定してインスタンスを生成し，それをバッファリング可能にする標準的な方法は次の通りです．

書き込みファイル指定

BufferedWriter bw = new BufferedWriter(new FileWriter(”test1.txt”));

読み込みファイル指定

BufferedReader br = new BufferedReader(new FileReader(”test.txt”));

これでファイルがオープンし，入出力の準備ができたことになります．このようにしてオープンしたファイルに対して，入出力処理を行います．処理が終了したら，bw.close(); と br.close(); でファイルを閉じます．

データの読み込みを行うには，Reader クラスのインスタンスメソッドに加え，文字ストリームから改行終端文字列を読み取る **readLine()** メソッドを用います．

データの書き込みを行うには，Writer クラスのインスタンスメソッドを用います．

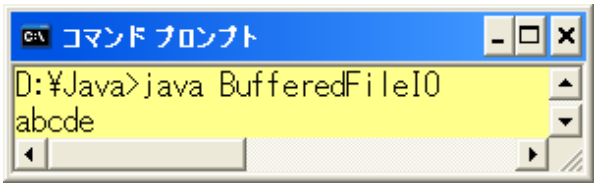
例題 11.4 バッファありファイル入出力

書き込みファイルを”test1.txt”とし，文字列”abcde”を書き込み，その後，書き込んだ文字列を読み取り表示し，ファイルを閉じるプログラムを作成しなさい．

```
=====================================================
import java.io.*;
class FileIO
{
        public static void main(String[] args)
        {
            String str = ”abcde”;
            try
            {
                //BufferedWriter オブジェクトの作成
                BufferedWriter bw = new BufferedWriter(new FileWriter(”test1.txt”));
                //ファイルに文字列を書き込む
                bw.write(str);
                bw.write(”¥r¥n”);
                //BufferedWriter オブジェクトを閉じる
                bw.close();
            }
            catch(Exception ex)
            {
                System.out.println(”Exception: ” + ex);
            }
            try
            {
                //BufferedReader オブジェクトの作成
                BufferedReader br = new BufferedReader(new FileReader(test1.txt”));
                //文字を読み取って表示
                String s;
                while((s = br.readLine()) != null)
                {
                    System.out.print(s);
                }
                //BufferedReader オブジェクトを閉じる
                br.close();
            }
            catch(Exception ex)
            {
                System.out.println(”Exception: ”+ ex);
            }
        }
}
```

==========

実行結果



バイトストリーム	
InputStream バイト入力用基本クラス	OutputStream バイト出力用基本クラス
FileInputStream ファイルからのバイト入力	FileOutputStream ファイルへのバイト出力
FilterInputStream 入力をフィルタに通す BufferedInputStream 入力をバッファに読み込む DataInputStream 入力を基本データ型に解釈	FilterOutputStream 出力をフィルタに通す BufferedOutputStream 出力をバッファに入れる DataOutputStream 基本データ型を出力 PrintStream 文字と型データの出力

第 12 章

アプレット

ここまでは，コンソールアプリケーションについて学んできました．ここからは，**アプレット (applet)** を作成して利用する方法を学習します．アプレットとは，Web ブラウザまたはアプレットビューアなどのツールで実行される小さなプログラムのことです．アプレットビューアは Java Development Kit(JDK) に付属しています．

この章では，**java.awt** パッケージのクラスを使って文字列，図形，イメージを表示する簡単なアプレットの作り方を学びます．このパッケージには，さまざまなカラーやフォントを扱うクラスも用意されています．

最初の Java アプレット

アプレットのソースプログラムは，アプレット対応形式にする必要があります．全てのアプレットの基本クラスは **java.applet.Applet** クラスです．この基本クラスを継承して，アプレットを作成するので，アプレットの基本スタイルは，次のようになります．

```
import java.applet.Applet;
import java.awt.Graphics;
public class アプレット名 extends Applet
{
    //メソッド
}
```

このソースプログラムをアプレット名.java という名前で保存します．ここで，**java.awt.Graphics** クラスがインポートされているのは，アプレットでは，一般に画面に文字列や図形などを表示するので，文字をここで，例えばアプレットで文字列を表示したい場合，メソッドは，

```
public void paint(Graphics g)
{
    g.drawString("表示したい文字列",20,100);
}
```

のようになります．ここで，20 と 100 は画面上の x と y 座標を表します．

このようにして作成したアプレットをコンソールアプリケーションと同様に javac でコンパイルすると，アプレット名.class というクラスファイルが作成されます．このクラスファイルは Web ブラウザ上で動作させることになります．したがって，Web ブラウザ上で表示させるための HTML ファイルが必要となります．HTML ファイルには最小限，以下の事柄を記述する必要があります．

Java アプレットを起動する HTML プログラム

```
<HTML>
<BODY>
<APPLET code=”アプレット名.class” WIDTH=250 HEIGHT=250>
</APPLET>
</BODY>
</HTML>
```

例題 12.1 最初の Java アプレット

”Hello World”と表示するアプレットを作成しなさい．

```
==========================================================
import java.applet.Applet;
import java.awt.Graphics;

public class GreetingApplet extends Applet
{
        public void paint(Graphics g)
        {
            g.drawString(”Hello World”, 50, 100);
        }
}

==========================================================
```

実行結果

このソースコードを詳しく見ていきます．

1 行目では **java.applet.Applet** クラス，2 行目では **java.awt.Graphics** クラスをインポートしています．**Applet** クラスは，すべてのアプレットの基本クラスです．**Graphics** クラスは，**AWT(Abstract Window ToolKit)** によって提供されます．4 行目では，Greetin g Applet クラスが Applet クラスから派生することを示しています．次に，**paint()** というメソッドを定義します．このメソッドの役目は，アプレットの出力を生成することです．paint() は Graphics オブジェクトを引数として受け取ります．文字列を実際に表示するのは，Graphics オブジェクトの **drawString()** メソッドです．このメソッドの 1 つ目の引数は，表示する文字列です，2 つ目と 2 つ目の引数で，文字列の表示を開始する位置を指定します．

練習問題 12.1 g.drawLine(x1,y1,x2,y2); を用いて点 (10,10) と点 (100,100) を結ぶ線を描画するアプレットを作成しなさい．

Graphics クラス

上の例題で **java.awt.Graphics** クラスの機能を利用しました．ここでは，このクラスについてより詳しく学びます．

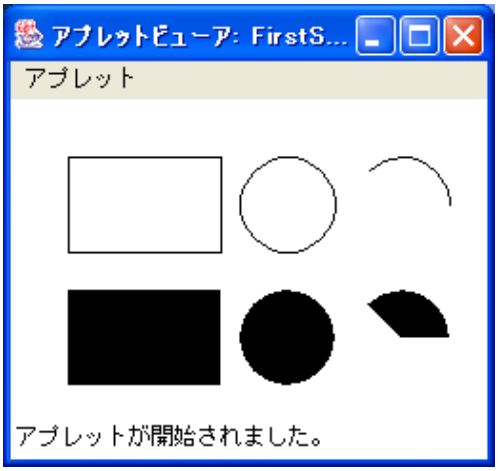
Graphics オブジェクトには，グラフィックスを出力するメソッド群がカプセル化されています．線 (Line)，楕円 (Oval)，四角形 (Rectangle)，多角形 (Polygon)，文字列 (String)，イメージ (Image)，文字 (Chars)，弧 (Arc) を描画することができます．

Graphics クラスの主なインスタンスメソッド	
void drawArc(int x, int y, int w, int h, int startAngle, int arcAngle)	座標 x,y に左上隅が配置されるように幅 w, 高さ h の矩形に接する円弧を描く startAngle で開始角度を指定し arcAngle で弧の展開角度を指定する
void drawImage(Image img, int x, int y, ImageObserver io)	座標 x,y に左上隅が配置されるように img を描画する．座標処理の進行状況は， io に送られる
void drawLine(int x0,int y0, int x1, int y1)	座標 x0,y0 と x1,y1 を結ぶ線を描画する
void drawOval(int x, int y, int w, int h)	座標 x,y に左上隅が配置される幅 w 高さ h の四角形に接する楕円を描く
void drawPolygon(int[] x, int[] y, int n)	n 個の頂点を持つ多角形を描画する． 頂点の座標は，配列 x と y の要素と して引き渡す．
void drawPolyline(int[] x, int[] y, int n)	n 個の頂点を持つ多角線を描画する． 頂点の座標は，配列 x と y の要素と して引き渡す．
void drawRect(int x, int y, int w, int h)	座標 x,y に左上隅が配置される幅 w， 高さ h の四角形を描画する
void drawString(String str, int x, int y)	str を座標 x,y に描画する
void fillArc(int x, int y, int w, int h, int startAngle, int arcAngle)	座標 x,y に左上隅が配置されるように幅 w, 高さ h の矩形に接する円弧を startAngle で開始角度を指定し arcAngle で弧の展開角度を指定し 塗りつぶす
void fillOval(int x, int y, int w, int h)	座標 x,y に左上隅が配置される幅 w 高さ h の四角形に接する楕円を塗りつぶす
void fillPolygon(int[] x, int[] y, int n)	n 個の頂点を持つ多角形を塗りつぶす． 頂点の座標は，配列 x と y の要素と して引き渡す．
void fillRect(int x, int y, int w, int h)	座標 x,y に左上隅が配置される幅 w， 高さ h の四角形を塗りつぶす
Color getColor()	現在のオブジェクトのカラーを取得する
Font getFont()	現在のオブジェクトのフォントを取得する
FontMetrics getFontMetrics()	現在のオブジェクトのフォント メトリックスを取得
void setColor(Color c)	グラフィックコンテキストの現在のカラー として c を設定
void setFont(Font f)	現在のオブジェクトのフォントとして f を 設定

例題 12.2 Graphics クラス
座標 30,30 に幅 80 高さ 50 の四角形，座標 120,30 に長軸 50，短軸 50 の楕円，座標 180,30 に幅 50，高さ 50 に接する 0〜135°の弧およびこれらを　y 軸方向に 70 平行移動し塗りつぶした図形を描画するアプレットを作成しなさい．

```
=====================================================
import java.applet.Applet;
import java.awt.Graphics;

public class FirstShapes extends Applet
{
        public void paint(Graphics g)
        {
            g.drawRect(30,30,80,50);
            g.drawOval(120,30,50,50);
            g.drawArc(180,30,50,50,0,135);
            g.fillRect(30,100,80,50);
            g.fillOval(120,100,50,50);
            g.fillArc(180,100,50,50,0,135);
        }
}

=====================================================
```

実行結果



Color クラス

java.awt.Color クラスを用いて描画と背景に色を使うことができます．使用頻度の高い色は Color オブジェクトとしてフィールド値が用意してあります．次の 13 色です．

black	blue	cyan	darkGray
gray	green	lightGray	magenta
orange	pink	red	white
yellow			

これらを用いるには

```
g.setColor(Color.blue);
```

のように用います．

次に，赤，緑，青の 3 色を混合させた RGB モデルで色をしてすることができます．

```
g.setColor(new Color(255,0,0);
```

で赤色を指定することができます．

Color オブジェクトには，色に関するメソッド群がカプセル化されています．

Color クラスの主なインスタンスメソッド

static int HSBtoRGB(float h, float s, float b)	h,s,b で指定された色相，彩度，明度を int 型の値にコード化して返す
static int RGBtoHSB(float h, float s, float b)	r,g,b で指定された red,green,blue の要素からなるカラーを色相，彩度，明度の float 型配列に変換して返す
Color brighter()	現在のオブジェクトの色を明るくしたオブジェクトを返す
Color darker()	現在のオブジェクトの色を暗くしたオブジェクトを返す
int getBlue()	現在のオブジェクトのブルー値を返す
int getGreen()	現在のオブジェクトのグリーン値を返す
int getRGB()	現在のオブジェクトの RGB 値を返す
int getRed()	現在のオブジェクトのレッド値を返す

練習問題 12.2 座標 30,30 に幅 80 高さ 50 の四角形，座標 120,30 に長軸 50，短軸 50 の楕円，座標 180,30 に幅 50，高さ 50 に接する 0〜135°の弧およびこれらを　y 軸方向に 70 平行移動し青く塗りつぶした図形を描画するアプレットを作成しなさい．

テキストの表示

drawString() メソッドを用いて文字列を表示する方法はすでに学びました．