Java 入門

東海大学総合情報センター

第 1 版

■ 目次

■ 1章 Javaとは？

Javaは、1995年にSun Microsystemsから発表されたオブジェクト指向[1]型プログラミング言語です。実は単にJavaと言った場合、それはプログラミング言語だけではなく、Javaに関連した技術全体を意味するといった方が正しいかも知れません。実際に、Javaはパソコンで動作するだけでなく、携帯電話などでも使用されています。しかし、このテキストでは「プログラミング言語としてのJava」を取り上げています。

1.1 Javaの特徴

Javaは、他の言語と比較して、次のような特徴があります。

(1) 様々なOS[2]に対応している

どんなOS上でも、Javaが実行できる環境があれば動作する。この点は現在のところ完全ではないが、"WriteOnce,RunAnywhere.（一度プログラムを作成すれば、どのプラットフォームでも実行できる）"ということが目標とされている。

(2) Webページ上で実行可能

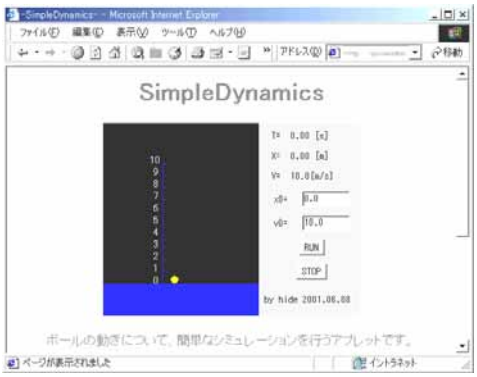
Web ブラウザ上で実行可能であり、インターネットを通じてプログラムを配布･実行することができる。開発者はプログラムの発信元のファイルだけ管理しておけば、利用者は常に最新のものを使うことができるようになる。

(3) 文法がC/C++に似ている

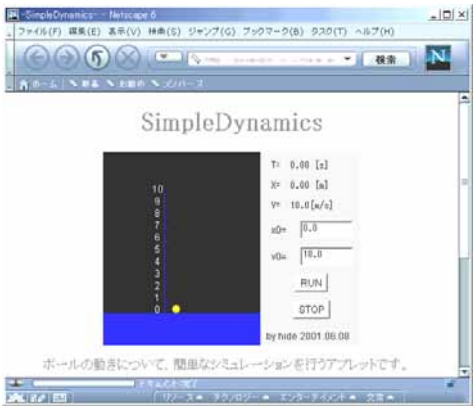
CやC++でプログラムを組んだことがある人には理解しやすい。しかも、ポインタや構造体がないなど、言語仕様がC/C++よりシンプルになっている。

これ以外にも、コンパイル[3]時の厳しいチェック（安定性･セキュリティ向上）、ネットワークライブラリを標準で装備、マルチスレッド[4]機能をサポートなどの特徴があります。

先に挙げた特徴(1)(2)の具体例として、様々なOSのWebブラウザ上で同じアプレットを実行したときの画面をいくつか示します。



Windows2000&InternetExplorerの場合



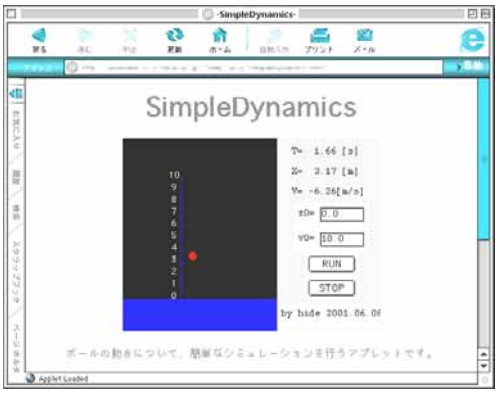
Windows2000&NetscapeNavigatorrの場合

[1]プログラミングにおいて、「全てのものはオブジェクトである。」という概念を取り込んだもの。プログラム内で使われている処理や変数は全てオブジェクトのメソッドまたはプロパティとして表わされ、オブジェクト同士のやり取り(メッセージ)でプログラムが実行されていく。

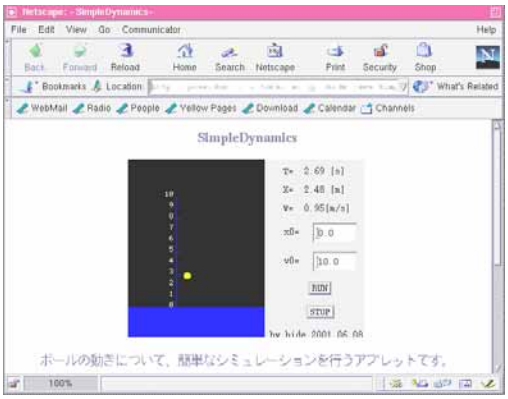
[2] Operating Systemの略。基本ソフトともいわれる。メモリやハードディスク、各種周辺機器などの管理や、ユーザインターフェースの提供など、コンピュータが動作する上で必要不可欠なもの。

[3] プログラミング言語を用いて記述したファイルをコンピュータで実行可能なファイルに変換すること。

[4] スレッドとはプログラムの実行単位を表し、マルチスレッドとは複数の実行単位を同時処理できることを意味する。



MacOS9&InternetExplorer の場合



UNIX(X-Window)&NetscapeNavigator の場合

このように、同じプログラムを異なる OS 上で実行できるということは、他のプログラミング言語で問題となる「OS が異なると実行できない」「同じプログラミング言語でも OS が異なるとソースファイルを書き換えなければならない」ということを考慮する必要がないということです。多くの OS で実行できるということは、それだけ開発に携わる人、利用する人が多くなるということです。また、最近では、携帯電話で Java のプログラムが実行できるようになり、Java が利用される世界はますます広がっています。このようなことから、Java が更なる可能性を秘めていることがわかります。

次に(3)の例として、1 から 100 までの合計を求めて表示するプログラムを、Java と C で記述した例を見てみましょう。

Java **プログラム例**	C **プログラム例**

Class Sum{
  public static void main(String args[]){
  int n=0;
  for(int i=1 ; i<=100 ; i++){
  n += i;
  }
  System.out.println("Answer:" + n);
  }
}
	#include<stdio.h>
	void main(){
	  int i,n=0;
	  for(i=1 ; i<=100 ; i++){
	  n += i;
	  }
	  printf("Answer:%d¥n", n);
	}

2 つのプログラムを比べてみると、Java と C は記述がよく似ていることが分かります。「似ている」ことを良い意味で考えると、「C を知っている人は、すでに覚えていることをそのまま Java で使うことができる」ということになります。また、Java が初めて学習するプログラミング言語となる人にとっては、Java を覚えた後に C 言語を覚えやすいということになります。ただし、「似ている」ことを悪い意味で考えると、「両者を混同してしまう」かもしれないことを覚えておいてください。

Java を実行する環境は、大まかに次の 2 種類があります。

- アプレット
  Java 対応の Web ブラウザ上で動作するプログラム。Web ブラウザとしてよく使われている Internet Explorer や Netscape Navigator では実行可能だが、対応している Java のバージョンに注意する必要がある。
- アプリケーション
  スタンドアローンで動作するプログラム。更に細かく分けると、実行するとウィンドウが表示されるウィンドウアプリケーション（GUI[5]）と端末上に文字を表示するコンソールアプリケーション（CUI[6]）の 2 種類がある。

プログラムを作成するときにどちらの環境を選ぶかは、使用目的により変わります。このテキストの前半では Java の基本的な文法を理解するためにコンソールアプリケーションを作成し、後半では比較的簡単なアプレットやアプリケーションを作成します。

1.2 Java と JavaScript との違い

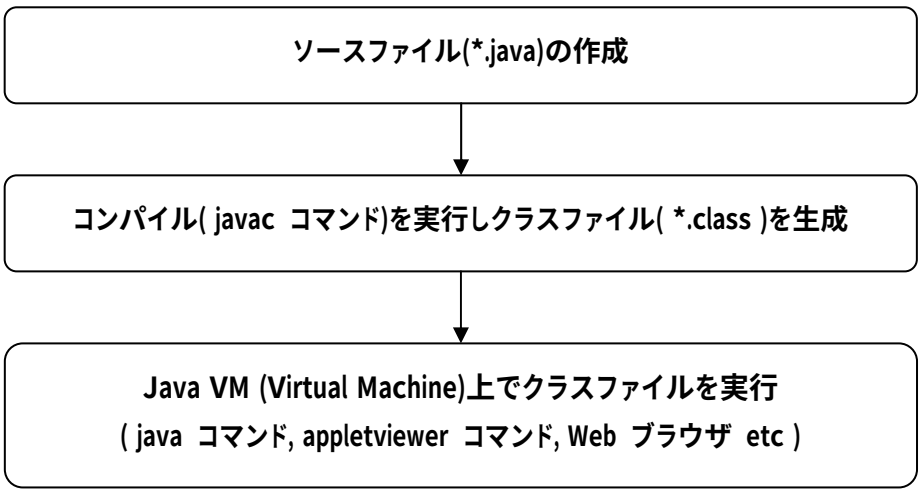
Java アプレットは Web ブラウザ上で実行することができますが、同じく Web ブラウザ上で実行可能なプログラミング言語に JavaScript が挙げられます。Java と JavaScript は言葉が似ているため、JavaScript は Java の機能限定版であると誤解している人も少なくないようです。実際には、この 2 つのプログラミング言語は全く異なるものです。

JavaScript は、Netscape Communications が開発した Web ブラウザ上で実行可能なプログラミング言語です。ごく初期の頃は LiveScript と呼ばれていましたが、Sun Microsystems との合意の下で JavaScript に名称変更されました。JavaScript は Web ブラウザの機能を拡張する目的で開発されており、Web ブラウザに関する制御や HTML[7]に関する操作を簡単に行うことができます。しかし、逆に考えるとそれ以外のことはできません。例えば、プログラム内で計算した結果をグラフやアニメーションとして表示したり、ネットワークを使ってデータベースを検索した結果を表示したりということはできません。JavaScript は、あくまでも Web ページでの表現を拡張するための言語でしかないということです。ただし、この目的で使用する場合には、最も適した言語といえます。

[5] Graphical User Interface の略。プログラムとユーザとの対話に、ボタンやアイコンなど視覚的な部品を使用しているもの。Windows の一般的なソフトウェアは、GUI で構成されている。
[6] Character User Interface の略。プログラムとユーザとの対話を文字のみで行うもの。Windows の DOS プロンプトやコマンドプロンプトでの操作のように、コマンドを文字で入力し、その結果も文字で表示される。
[7] HyperText Mark-up Language の略。ハイパーテキストを記述するための言語であり、Web ページはこれで構成されている。

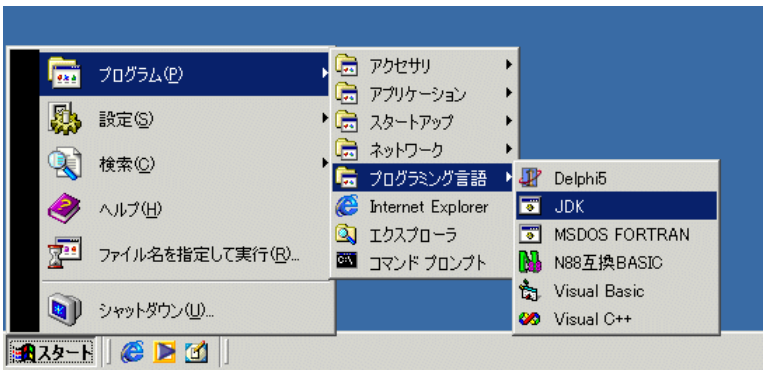
■ 2 章　プログラミングの流れ

Java のプログラミングは、以下の手順にしたがって行います。



初めて聞く言葉があるかもしれませんが、それぞれの項目は特に難しいことを行うわけではありません。プログラミングを繰り返して行ううちに慣れていくでしょう。まずは、プログラミング作業に先立ち、始めに Java の開発環境を起動させます。

[スタート]→[プログラム]→[プログラミング言語]→[JDK] と選択して、コマンドプロンプトを起動させてください。

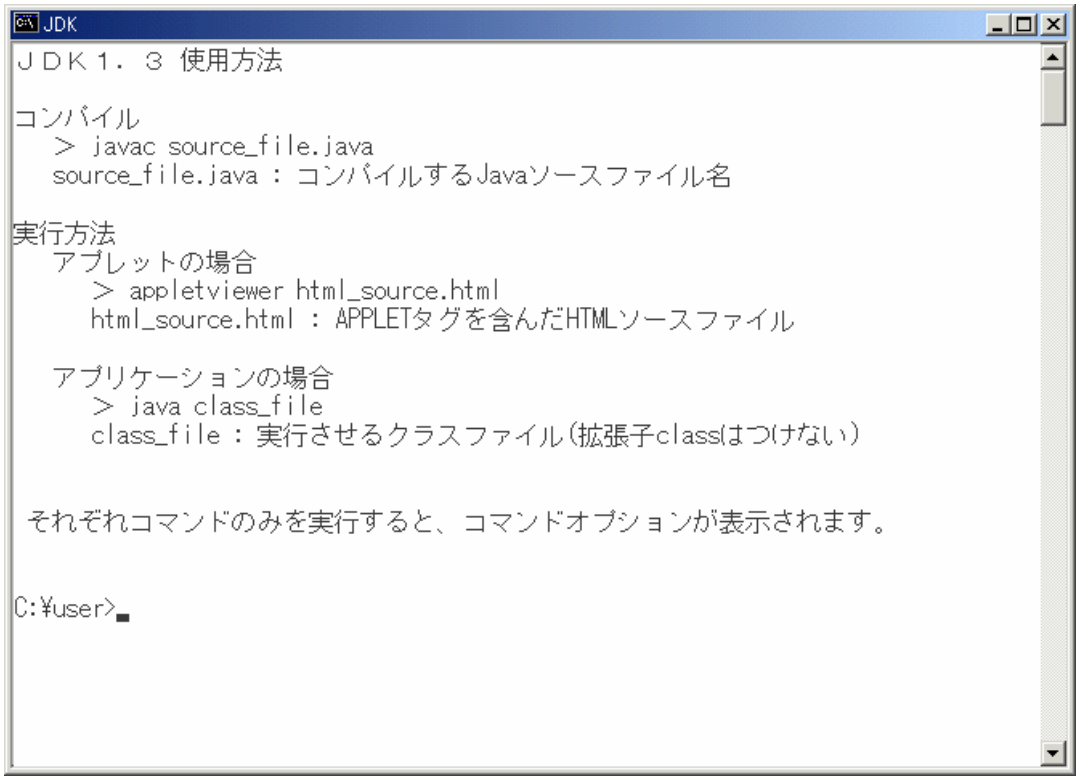


コマンドプロンプトとは、キーボードからコマンドを入力して処理を行うものです。行頭にある”c:¥user>”はプロンプトと呼びます。”c:¥user”の部分は、コマンドが実行されるときのディレクトリ（フォルダ）[8]を表しています。フロッピーディスクや Z ドライブを作業するディレクトリに変更する場合は、”>”に続けて”a: [Enter]”または”z: [Enter]”と入力してください。

2.1 ソースファイルの作成

Java でプログラムを作成する場合、まずソースファイル（source file）を用意する必要があります。ソースファイルは、コンピュータにプログラムの処理内容を伝えるために、人が理解できるような言語で書かれています。日本語や英語などの言語に文法があるように、プログラミング言語にも

[8] ファイルを分類して保存するために付けられるまとめられた単位。



Java, C, FORTRAN など言語があり、それぞれ文法が異なっています。

ソースファイルの作成には、メモ帳や EmEditor などのテキストエディタを使用します。ここでは”>”記号に続けて次のコマンドを入力してください。

```
C:¥user> notepad Sample.java
```

全ての文字を入力したら、[Enter]キーを押します。すると、メモ帳が起動します。コマンドプロンプトで文字の入力を間違えた場合には、カーソルキーの左右[←][→]や[BackSpace]キー、[Delete]キーなどを使って直すことができます。上のコマンドで、”Sample.java”がこれから作成するソースファイルの名前になります。ファイル名に含まれている".java"の部分は、拡張子[9]と呼ばれ、このファイルが Java のソースファイルであることを表します。

メモ帳が起動すると、「ファイル Smaple.java が見つかりません。新しく作成しますか？」と聞かれるので、[はい(Y)]をクリックします。そして、次に示されている例のとおりにプログラムを記述してください。

行	ソースファイル（Sample.java）
1	class Sample{
2	public static void main(String args[]){
3	System.out.println("This is a sample program.");
4	}
5	}

[9] ファイル名の末尾に付けられる”.(ドット)”で区切られた 3 文字前後の文字列で、ファイルの種類などを表している。

このプログラムは、画面に"This is a sample program."と表示するものです。もちろん" "で囲まれている文字列は、何でも構いません。なお、プログラムやファイル名の大文字・小文字は区別されるので、注意してください。

なお、メモ帳の起動をスタートメニューから選んで行ってもよいのですが、その場合は"Sample.java"というファイル名で保存することと保存先（コマンドプロンプトで表示されているディレクトリ）に注意してください。

2.2 コンパイルを実行しクラスファイルを生成

Java のソースファイルをコンパイルする場合には、javac コマンドを使用します。ここでは、次のコマンドを入力して、コンパイルを実行してください。

```
C:¥user> javac Sample.java
```

コンパイルを実行後、プロンプトだけが表示されれば、コンパイルが正しく行われたことになります。コンパイルの結果、ソースファイル中に記述してある class の名前に応じたファイルが、[クラス名].class という名前で作成されます。このファイルはクラスファイルと呼ばれています。dir コマンドで確認すると、Sample.class というファイルが作成されているはずです。

コンパイルの際に何かメッセージが表示された場合には、何らかのエラーまたはワーニング（警告）があることを意味します。このような場合には、ソースファイルを修正して、再度コンパイルを実行する必要があります。いくつかのメッセージについての説明がこの後にあるので、参考にしてください。 また、コマンドプロンプトでは、カーソルキーの上下[↑][↓]で過去に入力したコマンドを呼び出すことができます。

2.3 Java VM 上でクラスファイルを実行

クラスファイルの中身はバイトコードと呼ばれるもので構成されており、これを解釈して実際に処理を行うのは Java VM （Virtual Machine）です。実は、Java が様々な OS で実行可能なのは、この Java VM というソフトウェアが各 OS 毎に用意されていることによります。正常にコンパイルを実行できたら、次のコマンドを入力し、Java VM 上でクラスファイルを実行させます。

```
C:¥user> java Sample
```

ここで、注意しなければならないのは、クラスファイル名には".class"が付いていますが、実行させる際に指定する場合には".class"を省いた名前を指定するということです。

```
C:¥user> java Sample
This is a sample program.
```

"This is a sample program."と画面に表示されれば、正しく実行されたことになります。これが、Java によるプログラム作成の一連の流れです。この程度のプログラムならば、簡単にできることでしょう。しかし、実際にはコンパイルの際にエラーが発生して、ソースファイルを修正して再コンパイルしたり、実行できたものの満足のいく結果が得られないために、やはりソースファイルを修正して再コンパイルし再度実行したりの繰り返しで作成していくことになります。

なお巻末の付録に、コンパイル時や実行時に発生するエラーとそれぞれの対処方法についての説明がありますので、参考にしてください。

■ 3章 プログラム形式

3.1 ソースファイルの構造

前章で実行したプログラムは画面に文字を表示するだけという簡単なものでした。この程度の内容であれば「画面に文字を表示する」という命令を記述するだけでよさそうですが、実際には他にも記述しなければならないことがあります。この節では、プログラムに書かれている内容を具体的に説明していきます。

行	ソースファイル（Sample.java）
1	class Sample{
2	public static void main(String args[]){
3	System.out.println("This is a sample program.");
4	}
5	}

1行目：class Sample{

ここでは、このプログラムが"Sample"という名前のクラスであることを宣言しています。Java がオブジェクト指向型のプログラミング言語であることは第 1 章で紹介しましたが、実際には「クラス」という単位でまとめられたオブジェクトを扱っていきます。なかなか理解しにくいことですが、ソースファイルには必ずこの宣言が必要だということを覚えてください。 なお、クラスの定義は、”｛”～”｝”で囲まれる範囲で行います。したがって、最後の文字である”{“は、ここからクラスの定義が始まっていることを表しています。

2行目： public static void main(String args[]){

クラスを宣言したら、次にそのクラスで実行できるメソッドを定義します。ここでは Sample クラスのメソッドとして、”main”という名前のものを定義しています。「メソッド」は、実際に実行される処理が記述されているもので、C 言語では関数と呼ばれているものと同じです。こちらもクラスの定義と同様に、”｛”～”｝”で囲まれる範囲内に記述します。”public”,”static”,”void”などのキーワードは、それぞれに意味を持っているのですが、ここでは決まった書き方だと考えてください。

3行目： System.out.println("This is a sample program.");

main メソッドの処理として、“”で括られている文字を画面に表示させます。文末の”;”は、この命令文がここで終わりであることを表しています。

4行目： }

2 行目にあるメソッドの定義の始まりを表す"{"に対して、終わりを表す"}"です。

5行目：}

1 行目にあるクラスの定義の始まりを表す"{"に対して、終わりを表す"}"です。

以上のように、たった1行の文字を表示するだけでも、Javaのプログラムとして記述しなければならない事柄があります。1行目ではクラスの定義を行っていますが、Java のプログラミングとは、このように「クラス」を作成するということなのです。そしてクラスには、処理が記述された「メソッド」とクラスの特性を表した「データ[10]」と呼ばれるものを定義します。

メソッドとは「方法」や「機能」などを意味し、様々な計算や表示等、実際の処理内容を定義します。

データとは、そのクラスが持つ特性を数値や文字などで定義します。

これを踏まえて、Java のプログラムの構造を考えると、以下のようになります。

```
class クラス名{
    データの定義
            :
            :
    メソッドの定義( メソッドの引数 ){
            :
    }
            :
            :
}
```

例に挙げられているソースファイルでは、「Sample クラスに main メソッドが定義されている。（データの定義はない）」ということになります。

クラスとは、オブジェクト指向を理解する上で、非常に重要な概念です。ここで、分かりやすい例を挙げて説明しましょう。

例えば、「人」というクラスの定義を考えてみます。メソッドとデータには、それぞれ次のようなことが挙げられます。

メソッド：歩く，話す，投げる，道具を使う…

データ：名前，性別，年齢，身長，体重…

また、「車」というクラスの定義を考えてみると、次のようなことが挙げられます。

メソッド：走る，曲がる，止まる，ライトを点ける…

データ：名前，大きさ，乗車定員，スピード…

このような考え方をプログラミングに取り入れたのが、オブジェクト指向です。今は理解できなくても問題ありません。Java の学習を進めていくうちに、分かるようになるでしょう。

Java を使用して作成できるプログラム形式には、コンソールアプリケーション、アプレット[11]、アプリケーションの 3 種類があります。どの場合でも基本的なソースファイルの構造は同じです。それぞれの違いについて、次節以降で説明します。

[10] 「属性」または「プロパティ」と呼ばれることもある。

[11] 英語表記では”applet”となる。"let"は接尾語で「小さい」という意味を表すので、「小さなアプリケーション」を意味する。

3.2 コンソールアプリケーション

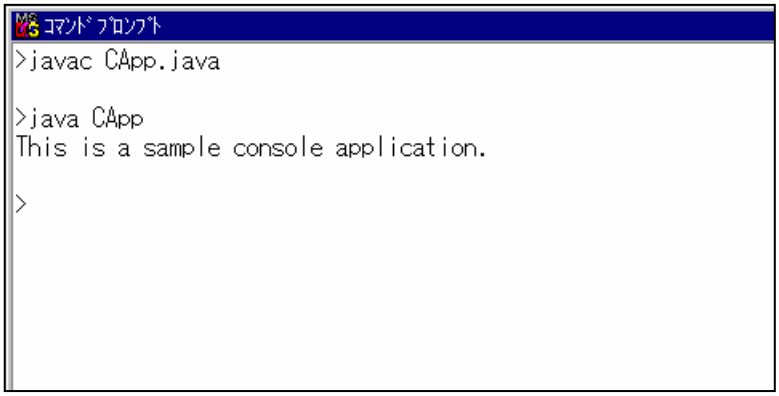
Java で作成できるプログラム形式として前節で例に挙げたものは、コンソールアプリケーションと呼ばれます。これは、コマンドを入力するようにプログラムを実行するタイプのものです。実行結果は文字で表示されます。

行	ソースファイル（CApp.java）
1	class CApp{
2	public static void main(String args[]){
3	System.out.println("This is a sample console application.");
4	}
5	}

実行方法

2 章で説明されている方法と同様です。上記のソースファイルをメモ帳などで作成し、コマンドプロンプトからコンパイルを実行します。エラー等がなければ、そのまま実行します。

実行結果



3.3 アプレット

Internet Explorer や Netscape Navigator などの Web ブラウザ上で実行するタイプのものです。また、Java の開発環境に含まれているアプレットビューワでも実行できます。なお、アプレットを実行する場合には、Java のプログラムの他に、Web ブラウザに読み込み可能な HTML が記述されたファイルも必要になります。

行	ソースファイル（Aplt.java）
1	import java.applet.Applet;
2	import java.awt.Graphics;
3	
4	public class Aplt extends Applet{
5	public void paint(Graphics g){

6	g.drawString("This is a sample applet.",20,20);
7	}
8	}

行	HTML ファイル（Aplt.html）
1	<APPLET CODE="Aplt.class" WIDTH="320" HEIGHT="60">
2	</APPLET>

実行方法

ソースファイルと HTML ファイルをメモ帳などで作成し、ソースファイルをコンパイルします。正常にコンパイルが終了したら、次のコマンドを入力してアプレットビューワを起動してアプレットを実行します。

```
C:¥user> appletviewer Aplt.html
```

アプレットの実行に、Internet Explorer や Netscape Navigator を使用しても構いません。この場合は、ソースファイルのコンパイル後、Web ブラウザを起動させて Aplt.html を開いてください。

実行結果

3.4 ウィンドウアプリケーション

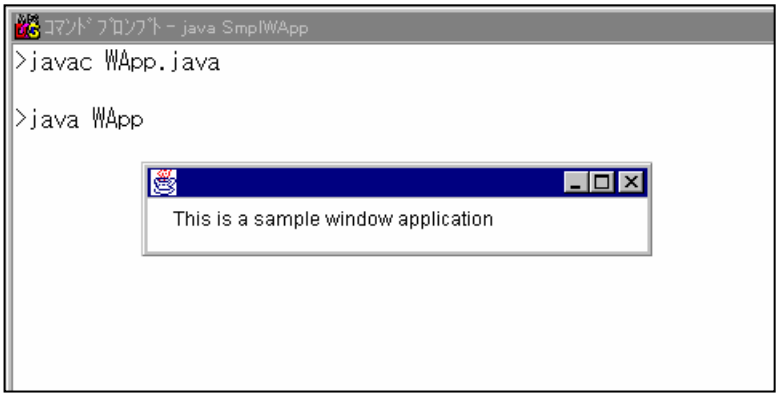
実行することによりウィンドウが表示され、ウィンドウ内で処理が行われるタイプのものです。普段使っている Windows のアプリケーションは、ウィンドウアプリケーションです。

行	ソースファイル（WApp.java）
1	import java.awt.*;
2	
3	class WApp extends Frame{
4	public WApp(){
5	setSize(320,60);
6	}
7	
8	public void paint(Graphics g){
9	g.drawString("This is a sample window application",20,40);
10	}
11	
12	public static void main(String args[]){
13	WApp wa = new WApp();
14	wa.show();
15	}
16	}

実行方法

コンソールアプリケーションを実行するときと同じ方法です。ソースファイルを作成し、コンパイルを行ってから実行します。終了させるためには、実行したコマンドを強制終了させます。コマンドプロンプトのウィンドウをクリックしてから、キーボードの[CTRL]キーを押しながら[C]を押してください。

実行結果



以上のように、3つの形式のプログラムを作成できるわけですが、それぞれ適材適所です。使用する目的に合った形式のものを選択するべきです。この講座では、Java の文法の学習のためにコンソールアプリケーションを用い、その後の発展としてアプレットを作成します。

■ 4章 Javaの文法

4.1 Javaの基本事項

ここでは、Java でプログラミングを行う際に覚える必要のある基本事項について説明します。次のプログラムに記述されている内容を例に取り上げます。このプログラムは、与えられた 2 つの整数について、それぞれ和・差・積・商（と余り）を計算し、表示するものです。

行	ソースファイル（Calc.java）
1	class Calc{
2	public static void main(String args[]){
3	// 変数の宣言
4	int i,j,s,d,p,q,r;
5	// 初期値の設定
6	i=7;
7	j=5;
8	// 四則演算
9	s = i + j;
10	d = i - j;
11	p = i * j;
12	q = i / j;
13	r = i % j;
14	// 結果の表示
15	System.out.println(i + " + " + j + " = " + s);
16	System.out.println(i + " - " + j + " = " + d);
17	System.out.println(i + " * " + j + " = " + p);
18	System.out.println(i + " / " + j + " = " + q + " ... " + r);
19	}
20	}

実行結果

```
C:¥user>javac Calc.java

C:¥user>java Calc
7 + 5 = 12
7 - 5 = 2
7 * 5 = 35
7 / 5 = 1 ... 2
```

変数とは？

例題では整数の計算を行っていますが、それぞれの計算を行うに当たって、「変数」を用いています。変数とは数学で出てくる"y=f(x)"の x や y のようなもので、いろいろな数値を取り得るものです。ソースファイルの 4 行目では、このプログラムで変数"i,j,s,d,p,q,r"を使用することを宣言しています。文頭の"int"は、これに続いて宣言されている変数が整数を扱うものであることを表しています。このように、Java ではプログラム中で使用する変数は必ず宣言しなければなりません。

変数の型

プログラムでは、数値や文字列、条件の判定などに変数を使います。変数には名前を付ける必要があるのと同時に、どんな内容の変数なのかを明示する必要があります。例題のプログラムの 4 行目では、"i,j,s,d,p,q,r"という名前の変数を整数型（int）として宣言されています。プログラム中で数値を扱う場合には、整数型と実数型があります。整数型は小数点以下の値がないもの、実数型は小数点以下の値を持つものです。Java で扱うことができる変数の内容とその宣言文、扱うことのできる値の範囲を次の表にまとめます。

型の名称		値の例	宣言文	値の範囲
整数	byte 整数型	123	byte	-128 ～ 127
	単整数型	123	short	-32768 ～ 32767
	整数型	123	int	-21×10^{8} ～ 21×10^{8}
	長整数型	123l	long	-922×10^{16} ～ 922×10^{16}
実数	単精度実数型	3.14159f	float	$\pm 1.4 \times 10^{-45}$ ～ $\pm 3.4 \times 10^{38}$
	倍精度実数型	3.14159d	double	$\pm 4.9 \times 10^{-324}$ ～ $\pm 1.7 \times 10^{308}$
boolean 型		true,false	boolean	true,false
文字型[12]		'a'	char	

プログラム中で単に数値を記述した場合、整数は整数型であるとみなされます。したがって、整数型が扱える範囲を超える数値を記述する場合は、数値の最後の"l（L:エル）"を付けて長整数型であることを示す必要があります。実数の場合、単に数値を書いた場合は倍精度実数型であるとみなされます。したがって、単精度実数型で宣言した変数に、数値を代入するときは、数値の最後に"f"を付ける必要があります。

また、整数同士の割り算を行うと、結果も整数型になるため、小数点以下の値は切り捨てられます。例題のプログラムで 7÷5 の計算結果である q の値が 1 になっているのはこのためです。

宣言文や命令文の記述

変数の宣言文や命令文は、文末に ；（セミコロン）を書き、文の終わりを明確にする必要があります。裏を返せば、各命令文の区切りは ；で判断されるため、1 行に複数の文を書くこともできます。したがって、ソースファイル中の 9~13 行目は、次のように 1 行で記述しても同じ意味になります。

[12] **「文字列（複数の文字）」を扱う場合は、String クラスを使用する。**

```
s = i + j; d = i - j; p = i * j; q = i / j; r = i % j;
```

状況に応じて、ソースファイルが見やすくなるような書き方を選択すればよいでしょう。

なお、宣言文・命令文が { } を伴うものの場合は、; を書く必要はありません。

識別子（クラス名，メソッド名，変数名, etc.）の命名

プログラム中の記述に使用する識別子は、次のルールに従って命名しなければなりません。例題のソースファイル中では、クラス名である"Calc"や、変数名"i,j,s,d,p,q,r"が識別子です。

- 1 文字目は、英字 A～Z,a～z、アンダースコア "_"、ドル記号 "$" でなければいけない。
- 2 文字目以降は、上の文字に加えて、数字 0～9 も使用できる。
- （後述の）予約語は使用できない。

なお、名前の文字数についての制限は特にありませんが、現実的な文字数で使用するようにしましょう。

予約語

次に示される単語は、予約語と呼ばれ、プログラム中で意味を持っている単語です。これらの単語は（前述の）識別子として使用できません。

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

算術演算子

演算子とは "=" , "+" , "-" , "*" , "/" などの記号のことで、大きく分けると変数の代入や四則演算などで使用される算術演算子と、値の大小や一致などの比較で使用される関係演算子（論理演算子）の 2 種類があります。例題では、6,7 行目で代入、9~13 行目で計算を行っています。Java で使用できる算術演算子には、次のようなものがあります。

演算子	処理	使用例	
=	代入	a = 22;	a の値は 22 になる
+	足し算	a = 22 + 8 ;	a の値は 30 になる
-	引き算	a = 22 – 8 ;	a の値は 14 になる
*	掛け算	a = 22 * 8 ;	a の値は 176 になる
/	割り算	a = 22 / 8 ;	a の値は 2 になる
%	割り算の余り	a = 22 % 8 ;	a の値は 6 になる
+=	和を代入	a += 22 ;	a=a+22 と同じ意味
-=	差を代入	a -= 22 ;	a=a-22 と同じ意味
*=	積を代入	a *= 22 ;	a=a*22 と同じ意味
/=	商を代入	a /= 22 ;	a=a/22 と同じ意味
++	インクリメント	a++ ;	a の値を処理してから 1 を足す
		++a ;	1 を足してから a の値を処理する
--	デクリメント	a-- ;	a の値を処理してから 1 を引く
		--a ;	1 を足してから a の値を処理する

ちなみに、"*"記号はアスタリスク、"/"記号はスラッシュと言います。

インクリメント・デクリメントは、演算子を変数の前に置くか後ろに置くかで意味が異なります。次に示す例を参考にしてください。

a = 8; b = a++ ;	a の値を b に代入してから、a の値に 1 を足します。 よって、実行後の a の値は 9、b の値は 8 です。
a = 8; b = ++a ;	a の値に 1 を足してから b に代入します。 よって、実行後は a の値も b の値も 9 です。

コメント文

ソースファイルの 3,5,8,14 行目には、"//"という文字の後に処理の説明が書かれています。これはコメント文といって、ソースファイル中に記述できるメモのようなものです。プログラムの実行には関係しませんが、自分以外の人がプログラムを読むときや自分が後からプログラムを読むときに理解しやすくするために使われるものです。なお、Java では、C 言語と同じように"/*"～"*/"で囲まれている範囲もコメント文と認識されます。

制御文

例題のプログラムには含まれていませんが、プログラミングを行う上で、ある条件に基づいて処理を切り替えることや反復して処理を行うということは必要不可欠です。これを行うためには制御文を使用します。制御文は、次の種類があります。

条件分岐	if – else switch – case

反復処理	for while do while
その他の制御文	break continue return

制御文については次節以降で説明します。

4.2 **条件分岐** (if – else)

与えられた条件を判断し、それに応じて処理を行います。文法は次のとおりです。

```
if ( 条件式 ) {
        処理          //「条件式」が成立する場合、「処理」を実行する。
}
```

```
if ( 条件式 ) {
        処理 1        //「条件式」が成立する場合、「処理 1」を実行する。
} else {
        処理 2        //「条件式」が成立しない場合は、「処理 2」を実行する。
}
```

```
if ( 条件式 1 ) {
        処理 1        //「条件式 1」が成立する場合は、「処理 1」を実行する。
} else if ( 条件式 2 ){
        処理 2        //「条件式 1」は成立しないが「条件式 2」は成立する場合は、
                      //「処理 2」を実行する。
} else {
        処理 3        //「条件式 1」「条件式 2」のどちらも成立しない場合は、
                      //「処理 3」を実行する。
}
```

if, else の後の処理が"{ }"で囲まれていますが、この囲まれている部分は、「ブロック」といいます。処理が複数の文にわたる場合にはブロック内に記述しますが、1 文の場合にはブロックにする必要はありません。これは、後述の制御文すべてに当てはまります。

条件式の部分には、関係演算子を使用して判断に用いる式を書きます。式が成立するかどうかは boolean 値 (true ,false) で表され、成立する場合は true、成立しない場合は false となります[13]。条件式で使用される関係演算子を次の表に示します。

[13] C言語のように、「不成立は 0、それ以外の数値は成立」とは判断されないので要注意。

演算子	判断	使用例	
==	一致	a==4	a の値が 4 の時 true
!=	不一致	a!=4	a の値が 4 以外のとき true
>	大小の比較	a>24	a の値が 24 より大きいとき true
<		a<24	a の値が 24 より小さいとき true
>=	大小の比較 (値を含む)	a>=24	a の値が 24 以上のとき true
<=		a<=24	a の値が 24 以下のとき true
&&	論理積	a==4 && b==24	a==4 と b==24 という条件が同時に成立するとき true
||	論理和	a==4 || b==24	a==4 と b==24 という条件のどちらかが成立するとき true

次に if – else 文を使用したプログラムの例として、与えられた数値が奇数か偶数かを判断するプログラムを示します。

行	ソースファイル（TestIf.java）
1	class TestIf{
2	public static void main(String args[]){
3	int i, j;
4	i = j = 0;
5	try {
6	i = Integer.parseInt(args[0]);
7	} catch(Exception e) {
8	System.err.println("**入力エラー**");
9	System.exit(1);
10	}
11	j = i % 2;
12	if (j == 0) {
13	System.out.println("**それは 偶数 です。**");
14	} else {
15	System.out.println("**それは 奇数 です。**");
16	}
17	}
18	}

実行結果

```
C:¥user>javac TestIf.java

C:¥user>java TestIf 11          （11 を入力する場合）
それは 奇数 です。
```

このプログラムでは、判断に用いる数値を実行時の引数として与えています。5～10 行目がこのための処理です。ここで使用されている try–catch(-finally)文については、「4．8 例外処理」の説明を参照してください。

4.3 条件分岐 (switch – case)

判断に使用する式の結果に応じて、処理を分けて行います。判断のために使用できる変数は、byte, short, int, char で宣言されたものです。文法は次のとおりです。

```
switch( 式 ) {
      case 値 1:
            処理 1        // 式の結果が「値 1」のときに「処理 1」を実行する。
            break;
      case 値 2:
            処理 2        // 式の結果が「値 2」のときに「処理 2」を実行する。
            break;
      (省略)
      case 値 n:
            処理 n        // 式の結果が「値 n」のときに「処理 n」を実行する。
            break;
      default:
            処理 x        // 式の結果が上記のどれとも一致しないときに
                          //「処理 x」を実行する。
}
```

なお、各 case 文の終わりに break 文を記述しないと、式の値に関わらずその後ろにある case 文の処理も実行されてしまいます。break 文は、処理をブロックの外側に移すことを意味します。

行	ソースファイル（TestSwitch.java）
1	class TestSwitch{
2	public static void main(String args[]){
3	int i = 0;
4	try {

```
5           i = Integer.parseInt( args[0] );
6       } catch(Exception e ) {
7           System.out.println( "入力エラー" );
8           System.exit(1);
9       }
10      switch ( i ) {
11          case 1:
12              System.out.println( "数字の 1 は工場の煙突です。" );
13              break;
14          case 2:
15              System.out.println( "数字の 2 はお池のあひるです。" );
16              break;
17          case 3:
18              System.out.println( "数字の 3 は赤ちゃんのお耳です。" );
19              break;
20          default:
21              System.out.println( "1 から 3 までの数字を指定してください。" );
22          }
23      }
24  }
```

実行結果

```
C:¥user>javac TestSwitch.java

C:¥user>java TestSwitch 2
数字の 2 はお池のあひるです。

C:¥user>java TestSwitch 5
1 から 3 までの数字を指定してください。
```

4.4 反復処理 (for)

繰り返して処理を実行する場合は、for 文を使用します。for 文の文法は、次のとおりです。

```
for ( 式 1 ; 式 2 ; 式 3 ){
    処理        //「条件式」が成立するときは、「処理」を実行する。
                //  不成立のときは、for ループから出る。
}
```

この命令では、まず式 1 を実行し、次に式 2 を評価します。結果が true であれば処理の部分が実行されます。実行後、式 3 を計算し、再度式 2 が評価されます。評価された結果が true であれば処理を実行するということを繰り返し、式 2 が false になるまでこれを行います。この説明から分かるとおり、式 2 は boolean 型の結果となるものである必要があります。具体的な使用方法については、サンプルプログラムを参照してください。

行	ソースファイル（TestFor.java）
1	class TestFor{
2	public static void main(String args[]){
3	int n, sum;
4	n = sum = 0;
5	try {
6	n = Integer.parseInt(args[0]);
7	} catch(Exception e) {
8	System.out.println("**入力エラー**");
9	System.exit(1);
10	}
11	for (int i=1 ; i<=n ; i++) {
12	sum += i;
13	}
14	System.out.println("1 **から**" +n+ "**までの合計は、**" + sum +"**です。**");
15	}
16	}

実行結果

```
C:¥user>javac TestFor.java

C:¥user>java TestFor 100
1 から 100 までの合計は、5050 です。
```

サンプルプログラムでは for 文の中で”int i=1”という宣言を行っていますが、これを見て「おや？」と思う人がいるかもしれません。C 言語や FORTRAN では、使用する変数の宣言はプログラムの先頭で行われます。これに対して Java では、変数の宣言をプログラム中の任意の場所で行えます。メソッドの定義の先頭で宣言された変数は、メソッド中のどこでも使用できますが、プログラムの途中で宣言された変数はブロックで囲まれている範囲でしか使用できません。したがって、for 文の中で宣言されている変数 i は、for 文のブロック内でのみ使用できることになります。

4.5 反復処理 (while)

条件式が成立している間、繰り返して処理を行います。文法は次のとおりです。

```
while ( 条件式 ) {
        処理          //「条件式」が成立するときは、「処理」を実行する。
                      // 不成立のときは、while ループから出る。
}
```

for 文とは異なり、while 文では条件式だけが評価されます。したがって、ループ内で条件式を満たす処理が行われないと、while ループの次に実行されるべき処理が実行されません。このような処理を「無限ループ」と呼びます。

行	ソースファイル（TestWhile.java）
1	class TestWhile{
2	public static void main(String args[]){
3	int n, sum;
4	n = sum = 0;
5	try {
6	n = Integer.parseInt(args[0]);
7	} catch(Exception e) {
8	System.out.println("入力エラー");
9	System.exit(1);
10	}
11	int i = 1;
12	while (i<=n) {
13	sum += i;
14	i++;
15	}
16	System.out.println("1 から " + n + " までの合計は、" + sum +" です。");
17	}
18	}

実行結果

```
C:¥user>javac TestWhile.java

C:¥user>java TestWhile 10
1 から 10 までの合計は、55 です。
```

4.6 反復処理 (do-while)

条件式が成立している間、繰り返して処理を行います。while と異なるのは、こちらは処理を行ってから条件式の判定を行うので、最低 1 回は実行されるということです。do-while 文の文法は、次のとおりです。なお、条件式の後ろに";"を置く必要があるので、注意してください。

```
do {
      処理                  //「処理」を実行する。
} while ( 条件式 ) ;      //「条件式」が成立するときは、再度処理を実行
```

行	ソースファイル（TestDo.java）
1	class TestDo{
2	public static void main(String args[]){
3	int n, sum;
4	n = sum = 0;
5	try {
6	n = Integer.parseInt(args[0]);
7	} catch(Exception e) {
8	System.out.println("入力エラー");
9	System.exit(1);
10	}
11	int i = 1;
12	do {
13	sum += i;
14	i++;
15	} while (i<=n) ;
16	System.out.println("1 から" + n + "までの合計は、" + sum +"です。");
17	}
18	}

実行結果

```
C:¥user>javac TestDo.java

C:¥user>java TestDo 50
1 から 50 までの合計は、1275 です。
```

while 文と do-while 文との処理の違いは、例に挙げたプログラムで 0 までの合計を求めようとしたときに表れます。

while 文の場合は、条件式を判断してからループ内の処理を行うため、0 までの合計を求める場合はループ内の処理が行われません。一方 do-while 文の場合は、ループ内の処理を行ってから条件式が判断されるため、0 までの合計を求める場合でも処理を 1 回行ってからループを抜けることになります。

```
C:¥user>java TestWhile 0
1 から 0 までの合計は、0 です。

C:¥user>java TestDo 0
1 から 0 までの合計は、1 です。
```

4.7 その他の制御文 (break, continue, return)

ここでは、これまでに説明した文以外の制御文を取り上げます。これらの文は、他の制御文と組み合わせて使用したり、メソッドの終了に使用したりします。

break 文

if-else 文,switch-case 文,for 文,while 文,do-while 文で、実行中のブロックまたはループから、ブロックの外に処理を移します。

```
…{
      処理 1
      break;      // 処理 2 は実行されず、処理 3 が実行される。
      処理 2
}
処理 3
```

行	ソースファイル（TestBreak.java）
1	class TestBreak{
2	public static void main(String args[]){
3	int n, sum;
4	n = sum = 0;
5	try {
6	n = Integer.parseInt(args[0]);
7	} catch(Exception e) {
8	System.out.println("入力エラー");
9	System.exit(1);
10	}
11	for (int i=1 ; i<=n ; i++) {
12	if (i>50) { // 50 までの合計で break

```
13                          n = 50;
14                          break;
15                    }
16                    sum += i;
17              }
18              System.out.println( "1 から"+ n +"までの合計は、"+ sum +"です。");
19        }
20  }
```

continue **文**

for 文,while 文,do-while 文で実行中のループから、それ以降の処理を行わずに次の繰り返し処理に移行します。

```
…{
      処理 1
      continue;  // 処理 2 は実行されず、再び処理 1 が実行される。
      処理 2
}
処理 3
:
```

行	ソースファイル（TestContinue.java）
1	class TestContinue{
2	public static void main(String args[]){
3	int n=0;
4	try {
5	n = Integer.parseInt(args[0]);
6	} catch(Exception e) {
7	System.out.println("入力エラー");
8	System.exit(1);
9	}
10	for (int i=1 ; i<=n ; i++) {
11	if (i%2 == 0) continue; // 偶数の場合 continue
12	System.out.println(i + "は、奇数です。");
13	}
14	}
15	}

return **文**

これまで例に挙げてきたプログラムでは、クラスの定義にメソッドが 1 つしかありませんでした。クラスに複数のメソッドを定義した場合、あるメソッドを呼出して処理した後、呼び出し元のメソッドに戻るときには return 文を使用します。このような手法は、C 言語の関数を同様です。呼び出されたメソッドがデータを呼び出し元に渡す必要がある場合は、返り値を指定します。返り値の型は、メソッドの宣言と合わせる必要があります。なお、main メソッドの型宣言にある”void”とは、返り値がないことを表しており、この場合 return 文は必要ありません。

```
型宣言 メソッド名 ( 引数 ) {
        処理
        return 返り値;    // メソッドの型が void 型の場合には必要ない。
}
```

行	ソースファイル（TestReturn.java）
1	class TestReturn{
2	public static void main(String args[]){
3	int n = 0;
4	try {
5	n = Integer.parseInt(args[0]);
6	} catch(Exception e) {
7	System.out.println("入力エラー");
8	System.exit(1);
9	}
10	System.out.println("1 から"+n+"までの合計は、"+calc(n)+"です。");
11	}
12	static int calc(int n){
13	int sum = 0;
14	for(int i=0 ; i<=n ; i++){
15	sum += i;
16	}
17	return sum;
18	}
19	}

4.8 **例外処理** (try-catch-finally)

プログラム中で想定されていない処理が発生した場合、通常はプログラムが異常終了し、その後の処理は行われません。しかし Java では、このような想定されていない処理が発生したとき、どのように処理するかを予め用意しておくことができます。これを例外処理と呼び、try – catch - finally 文で定義します。

```
try {
      処理 1       // 例外が発生する可能性のある処理
} catch( 例外クラス 例外オブジェクト ) {
      処理 2       // 例外が発生した場合の処理
} finally {
      処理 3       // 処理 1 が正常に実行されたかどうかに関わらず実行される処理
}
```

この章のサンプルプログラムでは数値の入力をプログラム実行時に引数として与えていますが、このときに例外が発生した場合、「入力エラー」と表示してプログラムの実行を中止しています。

また、想定される例外が複数ある場合、それぞれの例外にについてその後の処理を分けることができます。サンプルプログラムの場合に想定される例外としては、「引数が指定されていない場合」「数値以外が指定された場合」が考えられます。そこで、例外毎に適切なエラーメッセージを表示するように変更したプログラム例を以下に示します。

行	ソースファイル（TestTry.java）
1	class TestTry{
2	public static void main(String args[]){
3	int n, sum;
4	n = sum = 0;
5	try {
6	n = Integer.parseInt(args[0]);
7	} catch(ArrayIndexOutOfBoundsException e) {
8	System.out.println("第 1 引数を指定してください。");
9	System.exit(1);
10	} catch(NumberFormatException e) {
11	System.out.println("整数を指定してください。");
12	System.exit(1);
13	} catch(Exception e) {
14	System.out.println("その他の入力エラー");
15	System.exit(1);
16	}
17	for (int i=1 ; i<=n ; i++) {
18	sum += i;
19	}
20	System.out.println("1 から" + n + "までの合計は、" + sum +"です。");
21	}
22	}

実行結果

```
C:¥user>javac TestTry.java

C:¥user>java TestTry                    （引数を指定していない場合）
第 1 引数を指定してください。

C:¥user>java TestTry abc                （数字以外を指定した場合）
整数を指定してください。

C:¥user>java TestTry 10
1 から 10 までの合計は、55 です。
```

「引数が指定されていない場合」や「数値以外が指定された場合」には、それぞれ発生する例外オブジェクトが異なります。前者の場合 ArrayIndexOutOfBoundsException（配列のインデックスが範囲外）が、後者の場合は NumberFormatException（数値形式が異なる）が発生します。なお、3 番目には Exception の場合を定義していますが、これは前述の 2 つ以外の例外が発生した場合に実行されます。

4.9 練習問題

この章で説明した内容を踏まえて、次のプログラムを作成します。

問題　　入力された年が閏年かどうかを調べるプログラムを作成しましょう。
なお、閏年の条件は次のとおりです。
・4で割り切れる年を閏年とする
・ただし、100で割り切れる場合は除外する
・なお、400で割り切れる場合は閏年とする

ヒント

この問題について（プログラムを作成するのではなく）自分で考える場合、以下のプロセスを辿ることになります。

① 西暦何年について調べるのか
② その年は閏年の条件に当てはまるかどうか

そして、これをプログラムで処理する場合、いくつかのプロセスを付加する必要があり、次のようになります。

① 使用する変数の宣言
② 西暦何年について調べるのか（年の入力）

③ その年は閏年の条件に当てはまるかどうか

④ 結果の表示

更に②の部分は、どのようにして年を入力するかについて考えなくてはなりません。この章のサンプルプログラムのようにプログラム実行時に引数として与えることにすると、付加するべきプロセスが更に増えます。

① 使用する変数の宣言

② 入力された年は文字型なので整数型に変換

③ 変換時にエラーが発生した場合の処理

④ その年は閏年の条件に当てはまるかどうか

⑤ 結果の表示

以上のことを踏まえて、それぞれのプロセスについて Java の文法で記述することになります。このようにして作成されたプログラムは次のようになります。

行	ソースファイル（LeapYear.java）
1	class LeapYear{
2	// 閏年の条件
3	// (1)4 で割り切れる
4	// (2)100 で割り切れない
5	// (3)400 で割り切れる
6	public static void main(String args[]){
7	// ①使用する変数の宣言
8	int i=0,x,y,z;
9	try {
10	// ②入力された年は文字型なので整数型に変換
11	i = Integer.parseInt(args[0]);
12	} catch(ArrayIndexOutOfBoundsException e) {
13	// ③変換時にエラーが発生した場合の処理 その 1
14	System.out.println("第 1 引数を指定してください。");
15	System.exit(1);
16	} catch(NumberFormatException e) {
17	// ③変換時にエラーが発生した場合の処理 その 2
18	System.out.println("数字を指定してください。");
19	System.exit(1);
20	}
21	x = i % 4; // 年を 4 で割る
22	y = i % 100; // 年を 100 で割る
23	z = i % 400; // 年を 400 で割る
24	// ④その年は閏年の条件に当てはまるかどうか
25	if (x==0 && y!=0 || z==0) {

```
26                  // ⑤結果の表示 閏年の場合
27                  System.out.println( i + "年は、閏年です。" );
28              } else {
29                  // ⑤結果の表示 閏年でない場合
30                  System.out.println( i + "年は、閏年ではありません。" );
31              }
32          }
33  }
```

実行結果

```
C:¥user>javac LeapYear.java

C:¥user>java LeapYear 2001
2001 年は、閏年ではありません。

C:¥user>java LeapYear 2000
2000 年は、閏年です。
```

■ 5章　アプレットの作成

この章では、これまでに学習した内容の実践として、アプレットの作成を行います。アプレットを作成することにより、Web ページ内でプログラムを実行できます。ここでは、例題として現在の日時を表示するアプレットを取り上げます。実行すると、次のように表示されるものです。

アプレットの作成の流れは、これまで取り上げてきたコンソールアプリケーションの作成とほとんど同じですが、実行方法が異なります。また、ソースファイルの内容には、アプレット特有のものがあり、実行される処理を目的別に用意する必要があります。これまでに作成したコンソールアプリケーションでは、クラス内に main()メソッドのみ定義していましたが、アプレットの場合には、画面の描画を行う処理、変数の初期化などプログラムの最初に行う処理などを別々のメソッドに定義します。また、コンソールアプリケーションでは、クラスの作成を1から行っていましたが、アプレットの作成では、予め用意されている他のクラスの機能を使って新しくクラスを作成します[14]。したがって、オブジェクト指向に関する考え方を、多少は意識する必要が出てきます。

この章では日時を表示するアプレットを例に挙げ、まずは単に文字を表示するだけの簡単なアプレットを作成し、終わりには時計として使用できるアプレットを作成します。

5.1　日時表示アプレット Version.1

アプレット作成のはじめの一歩として、まずは画面に文字を表示するだけのものを作成します。このような簡単なアプレットでも、これまで作成してきたコンソールアプリケーションとは異なり、オブジェクト指向の考え方を意識する必要が出てきます。まずは、プログラムを作成し、各行の説明を参照してください。

(1) ソースファイルの作成

次のソースファイルを作成します。併せて各行の説明も参照してください。

行	ソースファイル（WhatTime.java）
1	import java.applet.*;
2	import java.awt.*;
3	import java.util.*;
4	
5	public class WhatTime extends Applet{
6	public void paint(Graphics g){

[14] これを「サブクラスを作成する」という。

```
7               String s;
8               Date d = new Date();
9               s = d.toString();
10              g.drawString( s, 10, 30 );
11          }
12  }
```

１行目：import java.applet.*;

２行目：import java.awt.*;

３行目：import java.util.*;

Java には多くの機能があり、それらはクラスライブラリと呼ばれています。また、このクラスライブラリは、パッケージという単位でまとめられています。このパッケージの中に含まれているクラス（ここでは「機能」と考えてよいでしょう）を使用するためには、import 文で宣言します。ここで示されている例では、java.util パッケージ、java.applet パッケージ、java.awt パッケージを使用することを宣言しています。それぞれのパッケージは、プログラム中の次のクラスを使用するために宣言されています。

java.applet ・・・ アプレットを作成するために Applet クラスを使用
java.awt ・・・ アプレットに文字列を描画するために Graphics クラスを使用
java.util ・・・ 現在日時を調べるために Date クラスを使用

５行目：public class WhatTime extends Applet{

このプログラムが"WhatTime"という名前のクラスであることを宣言しています。それに続く"extends Applet"は、このクラスが Applet（正確には java.applet.Applet）クラスを継承したクラスであることを意味しています。「継承」とは、その元となるクラス（スーパークラス）のメソッドやデータ（変数）を引き継いだクラス（サブクラス）を作成することです。これによりサブクラスでは、スーパークラスが持っている機能を再定義せずに使用できるようになります。ここで示されている例では Applet クラスを継承していますが、これはアプレットを作成するときに必ず必要です。

６行目：public void paint(Graphics g){

アプレットの作成では、この paint()メソッドが必ず必要になります。このメソッドはアプレットが表示されるとき（起動時や再描画時）に実行されるので、画面の描画処理を記述します。

７行目：String s;

表示する文字列のための文字列変数の宣言です。

８行目：Date d = new Date();

現在日時を調べるためには、Date クラスを使用します。この行では、Date クラスから現在日時を得るために、Date クラスのインスタンスを生成しています。インスタンスとは、定義されているクラスに従い、実際にプログラムで使用されるオブジェクトのことを表します。

クラスの定義には、そのクラスの機能や振る舞いが記述されていますが、これをプログラム中で使用するためには具体的にどのように使うのか？ということを決める必要があります。ここで示されている例では Date クラスのインスタンスとして、現在日時を使用するようにインスタンスを生成しています。この操作は、住民票の発行を受けるときの手続に似ています。その為には市役所に出向き、予め用意されている書類に必要事項を記入して窓口に提出します。このとき、用意されている書類を「クラス」とするならば、必要事項が記入された書類を作成することは「インスタンスを生成する」ということになります。窓口に提出する書類は必要事項が記入されたものでなくてはならないように、プログラム中でもインスタンスを使用して処理を行うのです。

９行目：s = d.toString();

Date クラスのインスタンスの toString()メソッドを実行して、現在日時が入っているオブジェクトを文字列に変換します。toString()メソッドは、元々Date クラスで定義されているため、Date クラスから生成されたインスタンスである d はこのメソッドを継承[15]しているので実行できるわけです。

１０行目：g.drawString(s, 10, 30);

Graphics クラスの drawString(文字列, x 位置, y 位置)メソッドで、現在日時を指定した場所に表示します。

１１行目：}

paint()メソッドの記述の終了を表します。

１２行目：}

WhatTime クラスの記述の終了を表します。

「3 章 ソースファイルの構造」で、「Java のプログラミングとはクラスを作成するということ」と説明しています。このプログラムでは、Applet クラス、Date クラス、Graphics クラスを使用して、新しいクラス（WhatTime クラス）を作成しているとも言えます。それぞれのクラスが持っている機能や使用できるメソッド等については、Java2 SDK のドキュメントに含まれている「Java2 プラットフォーム API 仕様」を参照してください。なお、このドキュメントは、Sun Microsystems の Web サーバからダウンロードできます[16]。Java でプログラミングを行う上で非常に重要なドキュメントなので、いつでも参照できるよう手元に置いておくと役立ちます。コンピュータ室では、総合情報センターのホームページ[17]から、「設定方法・利用方法・Q＆A」→「ソフトウエア・ツールの利用」→「Java2 SDK Standard Edition ドキュメント」とリンクを辿って参照できます。

(2) コンパイルを実行しクラスファイルを生成

このプログラムをコンパイルするためには、"javac"コマンドを使用します。上記のプログラムを"WhatTime.java"というファイル名で保存し、次のようにコマンドを入力します。

[15] なお、継承しているメソッドの処理を書き換える（再定義する）ことを、オーバーライドするという。
[16] http://jp.sun.com/software/java/index.html
[17] http://www.cc.u-tokai.ac.jp/

```
C:¥user>javac WhatTime.java
```

コンパイルを実行してエラー等が表示された場合には、プログラムが正しく入力されているか確かめてください。

(3) アプレットを実行

今回作成しているのはアプレットなので、Web ブラウザ上で実行できます。Web ブラウザで実行するためには Web ページを記述する言語 HTML (Hyper Text Markup Language) で HTML ファイルを作成する必要があります。HTML の説明については、ここでは省略しますが、今回作成しているアプレットを実行するために、次のような HTML ファイルを"WhatTime.html"という名前で作成してください。

行	HTML ファイル（WhatTime.html）
1	<APPLET CODE="WhatTime.class" WIDTH="320" HEIGHT="40">
2	</APPLET>

ここでは、アプレットを実行するために最低限必要なことしか書かれていません。アプレットを実行する Web ページに文章や画像などを表示させたい場合には、通常の Web ページと同じように HTML で記述し、アプレットを表示させたい部分に上の例にしたがってアプレットの実行に必要な記述を行います。

１行目：<APPLET CODE="WhatTime.class" WIDTH="320" HEIGHT="40">

HTML ファイルでは、タグと呼ばれる"<"と">"で囲まれた文字列を使って Web ページを構成しますが、アプレットを表示させる部分には APPLET タグを使用します。APPLET タグには、次のオプションを指定する必要があります。

CODE　　・・・ 実行するアプレットのクラスファイル名
WIDTH　　・・・ アプレットを表示する領域の幅
HEIGHT　　・・・ アプレットを表示する領域の高さ

２行目：</APPLET>

APPLET タグの終了を表します。

アプレットの実行は、作成した WhatTime.html ファイルを Web ブラウザで開くか、または JDK に付属している appletviewer コマンドで行います[18]。この場合、次のコマンドを実行します。

[18] appletviewer コマンドでアプレットを実行した場合、APPLET タグ以外の HTML タグの記述は無視される。

```
C:¥user>appletviewer WhatTime.html
```

なお、appletviewer コマンドでアプレットを実行した場合、APPLET タグ以外の記述は無視されます。

appletviewer コマンドで実行した例を示します。

5.2 日時表示アプレット Version.2

前節で作成したアプレットは白黒で表示されるものですが、これだけでは少々物足りない感じがします。Web ページでアプレットを実行するからには、背景や文字に色をつけたり、文字のフォントを変えたりして、見た目にこだわりたい人もいるでしょう。ここでは、前節で作成したアプレットを変更して、カラフルなものにします。

ソースファイルと追加した部分の説明は、次のようになります。

行	ソースファイル（WhatTime2.java）
1	import java.applet.*;
2	import java.awt.*;
3	import java.util.*;
4	
5	public class WhatTime2 extends Applet{
6	public void init(){
7	setBackground(Color.blue);
8	Font f = new Font("Serif", Font.BOLD, 14);
9	setFont(f);
10	}
11	
12	public void paint(Graphics g){
13	String s;
14	Date d = new Date();
15	s = d.toString();
16	g.setColor(Color.cyan);
17	g.drawString(s, 10, 30);
18	}
19	}

６行目：public void init(){

init()メソッドは、アプレットを作成する際に必要なわけではありませんが、アプレットの起動時に変数の初期化や処理の準備など実行させたい処理がある場合に記述します。ここでは、アプレットの背景色の設定と使用するフォントの設定を行っています。

７行目：setBackground(Color.blue);

アプレットの背景色を指定します。"Color.blue"は、Color クラスのオブジェクトで、その名の通り青色を表します。予め用意されている色には、次のものがあります。

Color.white	白
Color.lightGray	ライトグレイ
Color.gray	グレイ
Color.darkGray	ダークグレイ
Color.black	黒
Color.red	赤
Color.pink	ピンク
Color.orange	オレンジ
Color.yellow	黄
Color.green	緑
Color.magenta	マゼンタ
Color.cyan	シアン
Color.blue	青

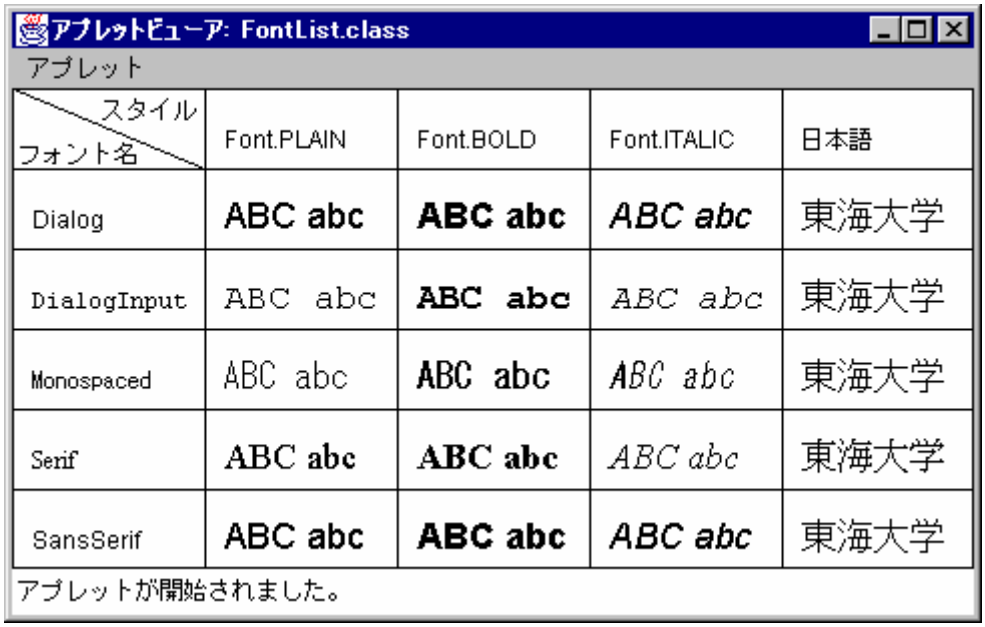
また、自分で任意の色を作成することも可能です。その場合は、次のように宣言します。

```
Color myColor = new Color( int r, int g, int b );
```

r, g, b には、0～255 の整数値を指定します。ここで作成された myColor を Color.blue の部分に指定します。

８行目：Font f = new Font("Serif", Font.BOLD, 14);

文字列の表示に使用するためのフォントの準備です。フォントを設定する場合には、フォントの名前、スタイル、大きさなどを指定し、Font クラスのインスタンスを生成します。使用できるフォントの名前とスタイルは次のとおりです。なお、これらのフォントを使用して日本語を表示させた場合、Serif は明朝体となり、それ以外はゴシック体となります。



９行目：setFont(f);

8行目では使用するフォントのインスタンスを作成しただけです。実際に使用するための設定はこの行で行っています。

１６行目：g.setColor(Color.cyan);

描画に使用する色を設定します。使用できる色は、7行目の説明にあるものと同じです。

ソースファイルの作成後、次の HTML ファイルを作成して実行します。

行	HTML ファイル（WhatTime2.html）
1	<APPLET CODE="WhatTime2.class" WIDTH="320" HEIGHT="40">
2	</APPLET>

実行結果は、次のようになります。背景色や文字色、フォントのサイズなどを、いろいろと変更してみてください。

アプレットで表示するデザインを変更した後に再度実行する場合、アプレットビューアーでは新しいアプレットが実行されますが、InternetExplorer や NetscapeNavigator では以前に読み込まれたアプレットが残ってしまい、新しいアプレットが実行されないことがあります。このような場合、次の操作をすることにより、新しいアプレットが実行されるようになります。

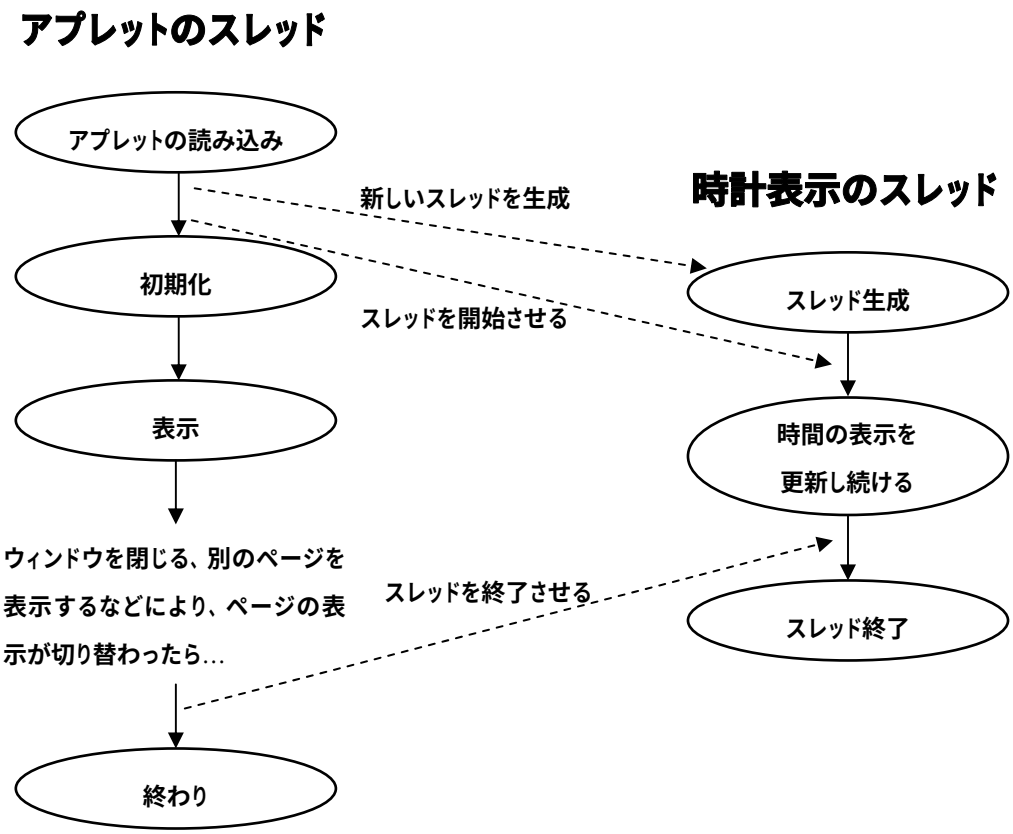
InternetExplorer の場合　　[CTRL] キーを押しながら [更新] ボタンをクリック
NetscapeNavigator の場合　[SHIFT] キーを押しながら [再読み込み] ボタンをクリック

5.3 日時表示アプレット Version.3

これまでに作ったアプレットは、実行されたときの日時を表示するだけのものでした。この節では、普通の時計と同じく時を刻むように変更します。

これまでに作成したアプレットでは実行されたときの日時が表示されるだけでしたが、時を刻むためには日時の表示を更新し続ける必要があります。これを実現するために、「スレッド」という機能を追加します。スレッドとは「分割された処理の単位」を表します。通常アプレットを実行する場合は、1 つのスレッドが処理されていることになりますが、ここでは時刻の表示を更新し続けるために別のスレッドを作成し、2 つのスレッドが並行して実行されるようにします。

これから作成する日時表示アプレットの処理の流れを次の図に示します。



これまでに作ったアプレットは、左側の「アプレットのスレッド」だけが実行されていたことになります。このように、複数のスレッドを並行して動作させることを「マルチスレッド」と言います。プログラムをマルチスレッドにすることで、時間の表示を更新しながら、他の処理を行うこともできるようになります。

このマルチスレッドを使用したプログラムは、次のようになります。

行	ソースファイル（WhatTime3.java）
1	import java.applet.*;
2	import java.awt.*;
3	import java.util.*;
4	
5	public class WhatTime3 extends Applet implements Runnable{
6	Thread trd;
7	
8	public void init(){
9	setBackground(Color.white);
10	Font f = new Font("Dialog", Font.BOLD, 12);
11	setFont(f);
12	}
13	
14	public void paint(Graphics g){
15	String s;
16	Date d = new Date();
17	s = d.toString();
18	g.setColor(Color.black);
19	g.drawString(s, 10, 30);
20	}
21	
22	public void run(){
23	while(trd != null){
24	try {
25	Thread.sleep(1000);
26	} catch (Exception e) {
27	System.out.println(e);
28	}
29	repaint();
30	}
31	}
32	
33	public void start(){
34	trd = new Thread(this);
35	trd.start();
36	}
37	
38	public void stop(){

```
39              trd = null;
40          }
41      }
```

追加された部分の説明は次のとおりです。

５行目：public class WhatTime3 extends Applet implements Runnable{

これまでのアプレットとは違い、"implements Runnable"という宣言が追加されています。これにより、スレッド機能を扱うことができます。ここで指定されている"Runnable"とは「インターフェース」の名前を表します。Java では、新しくクラスを作成する場合、"extends"宣言を使ってすでに存在しているクラスを継承できますが、このとき元となるクラスは１つしか指定できません。これを単一継承といいます。これに対して複数のクラスの機能を継承することを多重継承といいます[19]。Java では"Runnable"以外にも、マウスのボタンの状態を扱うための"MouseListener"、マウスの動きを扱うための"MouseMotionListener"など、様々なインターフェースが用意されています。インターフェースを利用する場合は、宣言したインターフェースに応じて定義しなければならないメソッドがあります。"Runnable"の場合には、"void run()"というメソッドを定義する必要があります。

６行目：Thread trd;

別のスレッドを作成するために、Thread クラスのオブジェクトを準備しています。これは、前述の図の「時計表示のスレッド」を扱うためのオブジェクトだと考えればよいでしょう。変数のように具体的に数値が代入されるわけではありませんが、スレッド自体を扱うために必要なものです。

２２行目：public void run(){

このメソッド内に記述した処理が、時計表示のスレッドとして実行されます。クラスの宣言で Runnable インターフェースを実装するよう指定した場合には、この run メソッドを必ず記述しなければなりません。

２３行目：while(trd != null){

そのまま読み取ると、trd が null でない場合に while ループが実行されることになります。trd は、6 行目で宣言されている Thread クラスのインスタンスで、null は値がないことを表します。したがって、trd に何らかの値が設定されないと、while ループは実行されないことになります。

２４行目：try {

25 行目にある Thread.sleep()を使用するためには、この try-catch 文が必要になります。この文では、try{ ～ }内で例外[20]が発生した場合、catch{ ～ }内に処理を移します。26 行目の説明も参照してください。

[19] Java と同じくオブジェクト指向言語である C++は、多重継承が可能である。
[20] 簡単にいうと、想定されていない処理や異常な処理のこと。

２５行目：Thread.sleep(1000);

時間の表示の更新を 1 秒ごとにするために、ここで処理を 1000 ミリ秒止めています。

２６行目：} catch (Exception e) {

24 行目の try に対応する catch です。try{ ～ }に書かれている処理は、処理を 1 秒止めるというものなので、この間処理が中断されたときには、例外が報告[21]されます。この報告を受け、catch 文では発生した例外ごとにその後に行う処理を分けることができます。ここでは、”Exception”という例外が発生した際に、発生した例外オブジェクトを”e”で受け取って、次のブロックに記述されている処理を実行します。実は”Exception”は全ての例外を表しており、このように記述することによって、発生した例外の種類によらず処理を行うことができます。また、Exception には様々な種類があり、発生した例外ごとに処理を分けることができます。

２７行目：System.out.println(e);

26 行目で例外の報告を受け取った際、この行が実行されます。ここでは、26 行目で受け取った例外オブジェクト”e”を（アプレットではなく）画面に表示します。

２９行目：repaint();

画面の再描画を行います。実際には、アプレットを背景色で塗りつぶした後に paint()メソッドを実行するという処理を行います。

３３行目：public void start(){

アプレットの処理が開始されたときに呼び出されるメソッドです。具体的には、init()メソッドの後に処理されます。

３４行目：trd = new Thread(this);

6 行目で準備している Thread クラスのオブジェクト trd に対して、インスタンスを生成しています。trd を 6 行目で宣言しておくことにより、このクラスで定義されているどのメソッドからもこのオブジェクトを使用できるのです。

３５行目：trd.start();

時計の表示を更新するスレッドを開始させます。これにより、run()メソッドが実行されます。

３８行目：public void stop(){

アプレットの処理が終了されるときに呼び出されるメソッドです。ここで、時計の表示の更新を停止させます。

３９行目：trd = null;

run()メソッド内の while 文の説明にあるとおり、trd に null が設定されると while ループは処理

[21] Java の中では、「例外 InterruptedException がスロー（throw）される」と表現される。

されなくなります。これにより、repaint()メソッドも呼び出されなくなり、時計の表示が更新されなくなります。

ソースファイルの作成後、次の HTML ファイルを作成して実行します。

行	HTML ファイル（WhatTime3.html）
1	<APPLET CODE="WhatTime3.class" WIDTH="320" HEIGHT="40">
2	</APPLET>

紙面では表現できませんが、実行すると時を刻み続けます。

■ 付録　コンパイル時や実行時に発生するエラーと対処方法

コンパイル時や実行時に発生するエラーは、単に文字を打ち間違えただけというものから、ソースファイルの書き換えを勧められるようなものまで、様々です。このうち、代表的なものとその対処方法は次のとおりです。

⑷ コンパイルの際に表示されるメッセージ

コマンド名を間違えている場合

```
C:¥user> jabac Sample.java
'jabac' は、内部コマンドまたは外部コマンド、
操作可能なプログラムまたはバッチ ファイルとして認識されていません。
```

コンパイルを行うコマンドは"javac"です。上の例では、”jabac”となっているため、エラーが発生しています。正しいコマンドは、”javac Sample.java”です。

ソースファイル名を間違えている場合

```
C:¥user> javac Sanple.java
エラー: Sanple.java を読み込めません。
エラー 1 個
```

```
C:¥user> javac Sample.jaba
javac: Sample.jaba は無効な引数です。
使い方: javac <options> <source files>
使用可能なオプションには次のものがあります。
  -g                        すべてのデバッグ情報を生成する
  -g:none                   デバッグ情報を生成しない
  -g:{lines,vars,source}    いくつかのデバッグ情報だけを生成する
  -O                        最適化。デバッグが抑制されるか、クラスファイルが大きくなる
  -nowarn                   警告を発生させない
  -verbose                  コンパイラの動作についてメッセージを出力する
  -deprecation              推奨されない API が使用されているソースの位置を出力する
  -classpath <path>         ユーザクラスファイルを検索する位置を指定する
  -sourcepath <path>        入力ソースファイルを検索する位置を指定する
  -bootclasspath <path>     ブートストラップクラスファイルの位置を置き換える
  -extdirs <dirs>           インストール済み拡張機能の位置を置き換える
  -d <directory>            生成されたクラスファイルを格納する位置を指定する
  -encoding <encoding>      ソースファイルが使用する文字エンコーディングを指定する
  -target <release>         特定の VM バージョン用のクラスファイルを生成する
```

この例では、どちらもソースファイルの名前を間違えています。正しいファイル名は、"Sample.java"なので、"javac Sample.java"とコマンドを入力します。

```
C:¥user> javac Sample.java
Sample.java:3: シンボルを解釈処理できません。
シンボル: メソッド prjntln (java.lang.String)
位置    : java.io.PrintStream の クラス
        System.out.prjntln("This is a sample program.");
                   ^
エラー 1 個
```

ソースファイル中に間違いがある場合

このエラーメッセージは、ソースファイル Sample.java の 3 行目に書かれている prjntln という文字列が間違っていることを示しています。例の場合は、println が正しい文字列なので、ソースファイルを修正します。

推奨されない API[22] を使用している場合

```
C:¥user> javac Sample2.java
注: Sample2.java は推奨されない API を使用またはオーバーライドしています。
注: 詳細については、-deprecation オプションを指定して再コンパイルしてください。
```

Java は、これまでに何度もバージョンアップを重ねてきました。その結果、古いバージョンで使用できた API が、新しいバージョンでは変更されている場合もあります。したがって、古いバージョンの Java 用に作成されたソースファイルを新しいバージョンの Java でコンパイルすると、このようなメッセージが表示されることがあります。次に、メッセージにしたがい"-deprecation"というオプションを付けてコンパイルを行うと次のようなメッセージが表示されます。

```
C:¥user>javac -deprecation Sample2.java
Sample2.java:16: 警告: java.awt.Component の resize(int,int) は推奨されません。
                resize( 320, 60 );
                ^
警告 1 個
```

[22] Application Programers Interface の略だが、ここでは単にソースファイルに記述する「命令」のことだと考えてもよい。

このメッセージから、推奨されないAPIとは16行目に書かれている"resize(320,60)"であることが分かります。 ただし、このメッセージは警告であり、エラーではないので、クラスファイルは作成されており、実行することも可能です。

しかし、このような API は将来のバージョンでサポートされなくなる可能性があります。したがって、新しいバージョンの API で書き換えておいたほうが安心です。このような場合、警告されたAPIをどのAPIに書き換えればよいかを知るためには、Java2 SDK[23]のドキュメントに含まれている「Java2 プラットフォーム API 仕様」を調べる必要があります。例に示した"resize(int, int)"について、このドキュメントには次のように書かれています。

resize
public void resize(int width, int height)
推奨されていません。JDK バージョン 1.1 以降は、setSize(int,int) に置き換えられました。

したがって、"resize(320,60)"を"setSize(320,60)"と書き直せばよいことが分かります。Java2 SDK のドキュメントには、その他にも様々な情報があるので、参考にするとよいでしょう。

⑻ 実行の際に表示されるメッセージ

コマンド名を間違えている場合

```
C:¥user> jaba Sample
'jaba' は、内部コマンドまたは外部コマンド、
操作可能なプログラムまたはバッチ ファイルとして認識されていません。
```

実行の際使用するコマンドは"java"なので、”java Sample”とコマンドを入力しなければなりません。

クラスファイル名の指定を間違えている場合

```
C:¥user> java Sumple
Exception in thread "main" java.lang.NoClassDefFoundError: Sumple
```

```
C:¥user> java Sample.class
Exception in thread "main" java.lang.NoClassDefFoundError: Sample/class
```

```
C:¥user> java Sample.java
Exception in thread "main" java.lang.NoClassDefFoundError: Sample/java
```

[23] SDK とは、Software Development Kit（ソフトウェア開発キット）の略。Java の開発に必要なコマンドや文書が含まれている。

正しいクラスファイルの名前は"Sample.class"であり、java コマンドにクラスファイルを指定する場合は".class"を付ける必要はないので、正しいコマンドは"java Sample"です。

ソースファイル中に間違いがある場合

```
C:¥user> java Sample
Exception in thread "main" java.lang.NoSuchMethodError: main
```

例に示されているソースファイルをそのまま入力した場合は、このようなメッセージが表示されることはありませんが、自分で作成したプログラムを実行したときにこのようなメッセージが表示された場合は、次のことを確認してください。

例に示されているソースファイルの 2 行目の"public static void main(String args[])"は、決まっている宣言です。"String args[]"を省いて"public static void main( )"と記述したり、"public static"を省いて"void main(String args[])"と記述したりすると、上記のメッセージが表示されます。通常、ソースファイルの記述に間違いがある場合は、コンパイル時の文法チェックでエラーになりますが、クラスの名前などを間違えている場合には、文法上は問題ないのでエラーにはならないのです。

コンパイル時、実行時に表示されるエラーには、上で説明しているもの以外にもたくさんあります。始めは間違いを見つけるのが大変かもしれませんが、表示されたメッセージをよく読み、その意味を考えれば、対処方法が分かるようになります。