Chapter 3. プログラムの説明

３－１．プログラムの記述の仕方

３－１－１．命令型のプログラム

通常のプログラムは、ただ単に順番に上から下（１行中に複数の命令があれば、左から右）へ向かって、命令が実行されます。例えば、Javaのアプリケーションのソースプログラムの中で、次のような記述があったとしましょう。printは、改行しないで端末にメッセージを出力します。

```
System.out.print( "Once upon a time, " );
System.out.println( "a rabbit lived in the forest." );
System.out.println( "His name was Melo." );
```

この記述の部分を実行させると次のような出力を得ることになります。これは、上のプログラムの断片が上から下、あるいは左から右に向かって実行された結果なのです。

```
Once upon a time, a rabbit lived in the forest.
His name was Melo.
```

１つの命令のことを文（Statement）と呼んだり、ステップ（Step）と呼んだりします。命令を使っていろいろな記述ができます。例えば、擬似的な自然言語を使って、人に何かを頼んでみることを考えてみてください。たとえば、公園の中央にいるうさぎに餌をやってほしいときには、次のように段階的に記述しますね。これと同じで、コンピュータに与えるプログラムも、一つ一つ手順を追って記述していきます。

```
go to the center of the garden ;
find a rabbit ;
hold  her ;
bring her  to your home ;
give  her   a bit of lettace ;
```

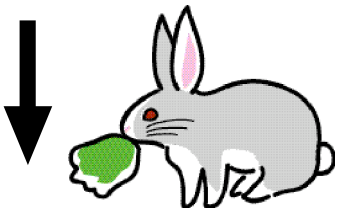

図 3-1　文の例

通常のプログラミング言語は、このような命令文の連なり（Sequence）が順番に実行されるので、命令型プログラミング言語（ImperativeProgrammingLanguage）と呼ばれています。Javaも基本的には、このような命令を羅列する形でプログラムを書き綴っていきます。なお、C言語やJavaの場合、通常１つの文の終わりには、ピリオドの代わりにセミコロン（;）をつけます。

プログラム中の命令文の列が実行されている状態のことを、プロセス（Process）と呼んでいます。１つのプログラムが実行されるということは、１つのプロセスが走り出すということと等しいのです。

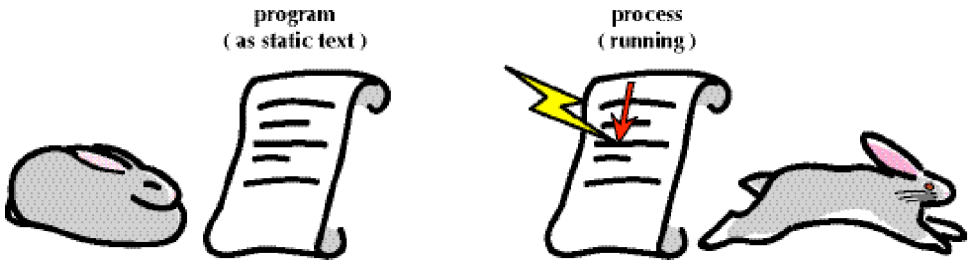

図 3-2　プロセスとプログラム

３－１－２．Javaの文について

コンピュータにさせたい処理の単位は、文（Statement）と呼ばれることは、既に紹介しました。Java言語でも文を幾つも記述することによって、仕事を達成させていきます。Java言語で書かれたプログラムを実行させると、記述された文を順番に上から一つずつ実行していくことになります。Java言語では文は、次のように最後に；をつけなければならないことをもう一度よく覚えておいて下さい。

▼Javaでの文の書式

```
処理させたいこと；
```

この書式を守っていけば、文を幾つも記述していくことができます。

▼アプリケーションで複数の文を記述していく

```
public static void  main(  .....  ) {
        最初にさせたい処理；
        次にさせたい処理；
        その次にさせたい処理；
                :
}
```

３－１－３．改行や空白と、コメント

Java言語では、記号と単語の間に空白や改行が入っても構いません。ただし、字下げ（Tabキーを使うとうまく行なえます）や書き方は、なるべく本書の書き方を見習ってください。美しい字下げがされていないプログラムは、それだけでも人に読む気をなくさせます。なお、ダブルクォーテーションマーク（”）で囲まれた文字列の中は、空白を空けるとそのまま画面に表示されますので注意してください。

コメントとは、コンパイラが無視する（読み流してくれる）記述のことです。後で、プログラムを他の人や自分がみるときに、どのような処理が書かれていたかを理解するための補助です。コメントには、何を書いても構いません。

コンパイラは次のように、/* から */ までで囲まれた部分をコメントとして取り扱います。この場合は、途中に改行が何行入っても構いません。

```
/*  コメントでございます */
```

あるいは、一行の途中で//という２つのスラッシュという記号を入れると、//以降から行末に掛かれている文字は、コメントとして取り扱われます。

```
//  コメントはプログラムを説明するために用いる
```

例えば作成したプログラムには、次の例のように毎回プログラムの先頭に、コメントを付けることわかりやすいでしょう。また、適宜コメントを挿入しておくのもよいでしょう。

```
//      プログラム名：〜〜をするプログラム
//      氏名：作成者の名前
//      作成完了：＊月＊日
```

```
public class Sample {
        public void main( String [ ] args ) {
                System.out.println( "Hello World!" );          // Hello World!と表示します
        }
}
```

３－２．クラスとオブジェクト

３－２－１．オブジェクト

オブジェクト指向モデル（Object Oriented Model）で、一番の中心となるのが、オブジェクト（Object）という概念です。オブジェクト指向モデルでは、命令が与える対象が明示されます。命令を与える対象って言えば、それはコンピュータに他ならないのですが、それを更に細分化して指示できるのです。命令を受け取る対象のことをオブジェクト（Object）と呼んでいます。今までのコンピュータのモデルでは、プログラムに従ってプロセスが１つ走り出し、それがオブジェクトという形で動いていきます。

オブジェクトには、いろいろな情報や機能を付加することができるのですが、そのような情報・機能群を指してオブジェクトのメンバ（Member）と呼んでいます。これらのオブジェクトのメンバの仕様は、クラス（Class）の中に記述されます。クラスは、オブジェクトは、このクラスの記述の元にして生成されます。なお、Javaのオブジェクト指向モデルでは、Smalltalk言語のモデルをかなり引き継いでおり、オブジェクト生成後もクラスとオブジェクトは密接な関係を維持しています。

３－２－２．クラスとオブジェクトの関係

オブジェクト指向モデルでは、既に生成されたオブジェクトは、同じクラスから生成されたもの同士がグループ化されています。クラスの中に、その種類のオブジェクトに関するメンバの特性（Property）が記述されているからです。メンバとして、次のような内容が記述されています。

・オブジェクトが実行できる機能を表したメソッドの定義
・オブジェクトが保持する情報を表したフィールド（Fields）

クラスにその特性が記述されているオブジェクトのことを特にインスタンス（Instance）と呼ぶこともあります。例えば、下の図ではうさぎ（Rabbit）のためのクラスがあり、このクラスには３つのメソッドが用意されています。eatCarrot、eatLettuce、jumpがそれにあたります。このクラスに属するオブジェクトには、Mary、Jane、Melody、しろという名前のオブジェクト（インスタンス）があります。

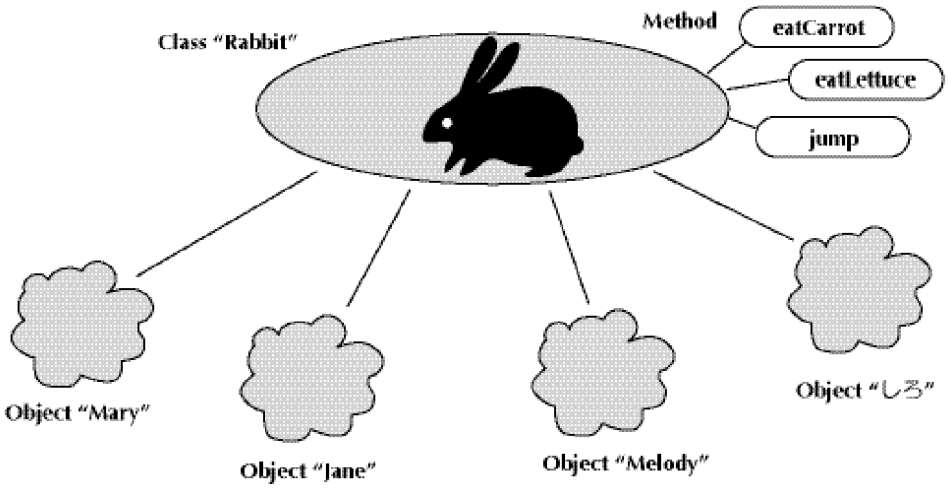

図3-8 クラスとオブジェクト

３－２－２．クラス間での継承について

さらにクラスについても、クラス間での階層（Hierarchy）があります。これは、ものごとを分類するときと同じで、大きな種類から、より小さな種類に分類されることを意味しています。クラス間の階層では、より上位のクラスが持っている特性は、下位のクラスに継承（Inheritance）されます。例えば、下の図では生物（Creature）というクラスの下には、植物（Plant）と動物（Animal）というクラスがあります。また、その下には猫（Cat）とうさぎ（Rabbit）というクラスがあります。

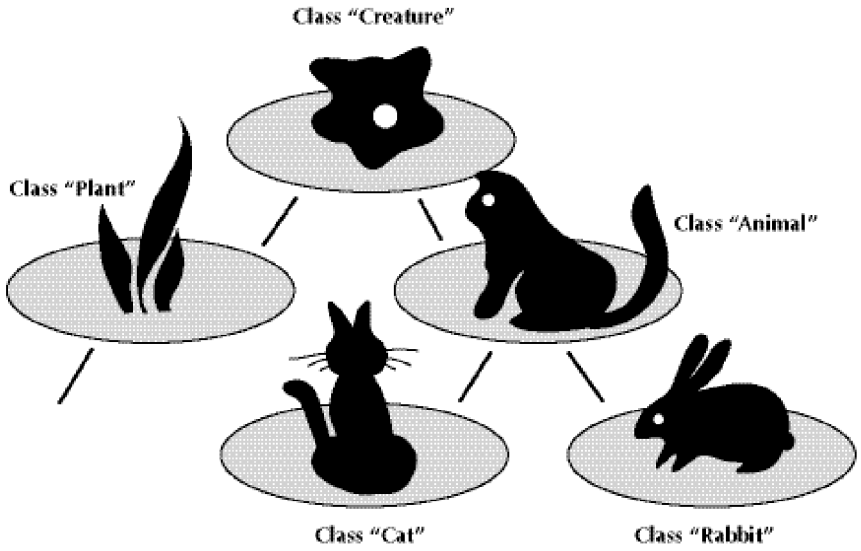

図3-9 クラス間の階層

上位のクラスが持っている特性が下位のクラスに継承される例を考えてみましょう。生物であれば、呼吸をしています。この特性は、動物にも植物にも引き継がれています。動物であれば、口を持っていたり、手足のようなものを持っています。この特性は猫やうさぎにも引き継がれています。

３－３．Javaでのクラスの定義

３－３－１．単体のクラスの定義

Java言語では、プログラムを１つのクラスとして記述します。プログラムを実行すると、このクラスが、プロセスとして走り始めるのです。具体的に、クラスに関してどのように記述するのかを紹介しましょう。最初に、Javaでは、単体でクラスを定義するのに、次のような書式を用います。

▼クラス定義の書式

```
class　クラス名 {
        クラスのメンバに関する記述
}
```

例えば、うさぎのためのクラスを一つ定義してみましょう。以下は、クラスRabbitを定義しています。....の部分は、何かそこに書いてあるという意味です。

```
class Rabbit {
        .....
}
```

３－３－２．継承を伴うクラスの定義

上位のクラスから特性を継承するときには、クラス定義のときに次のように記述します。継承については、後の節で説明します。上位のクラスは一つしか指定することができません。このため、Javaは単一継承と呼ばれています。

▼継承を伴うクラス定義の書式

```
class   クラス名 extends 上位のクラス {
          クラスの実体に関する記述
}
```

先ほどのうさぎ用のクラスを、動物のクラスを上位のクラスに指定して定義してみましょう。次のような記述になります。

```
class Rabbit extends Animal {
        .....
}
```

→クラス名は、　　Rabbit
→上位のクラス名は、　Animal

継承やインターフェースなど、Javaでは一つのクラスを記述するのにさまざまな機能が用意されています。グループ化して一つのクラスとして記述すれば、複数のオブジェクトで同じ特性を保持することができます。また、クラス継承を利用することにより、クラスの定義をする際に、上位のクラスの特性を利用する（もう一度定義しなくて済む）ことができます。

３－３－３．オブジェクトの生成

クラスはオブジェクトの雛型（Template）です。クラスを参考にして、オブジェクトが生成されることになります。逆に言えば、オブジェクトはどれかのクラスに属していなければなりません。あるクラスに属しているオブジェクトのことを、そのクラスのインスタンス（instance）と呼びます。

オブジェクト生成の書式

クラスから１つのオブジェクト（インスタンス）を生成するのは、次のような構文を用います。

▼書式：

```
new   クラス名( パラメータ );
```

たとえば、次のような形で記述します。FontクラスもColorもまだ紹介していませんが、AWTパッケージの中のクラスです。

```
new  Font( "Optima",  Font.PLAIN, 12 );      // Optimaという名前のフォントを作成する
new  Color(  244, 33, 111 );                 // RGB値が、244, 33, 111のカラーを生成する
```

このnewを伴って現れるクラス名に括弧がついたものは、メソッドの一種です。ですから、この書式はメソッド呼出しの書式になっています。この種類のメソッドを、コンストラクタ（Constructor）と呼んでいます。オブジェクトを生成するための特殊なメソッドなのです。通常のメソッド呼び出しと同じように、オブジェクト生成時に与えるパラメータを丸括弧の中に記述しても構いません。このパラメータは、生成されるオブジェクトのプロパティ（特徴）を指定するために用いられます。どのようなパラメータを必要とするかは、クラスによって異なります。例のようにパラメータが複数あれば、カンマで区切ります。また、パラメータが必要ない場合は、丸括弧内には何も記述しません。

３－４．メソッド

３－４－１．メッセージの伝達とメソッド

あるオブジェクトから別のオブジェクトに対して送られる命令のことをメッセージ（Message）と呼びます。

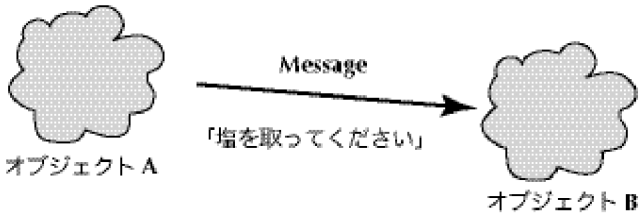

図3-4 オブジェクト間でのメッセージの送受

メッセージの受け手側（Receiver）のオブジェクトは、受け取ったメッセージに対応して何をすべきかということが予め記述されていなければならないのです。これは、プログラムが予め実行される前に記述されていなければならないことと同じだと思ってください。この記述がオブジェクト指向におけるプログラムなのである。この記述のことをメソッド（Method）と呼んでいます。以下の疑似プログラムは、メソッドを記述してみた例です。Javaでは丸括弧（）は、メソッドであることを示します。

```
passMe( salt  ) {   // when object receives "passMe salt" message
        take  salt   in   the  dining ;
        pass  salt   to   my   client  ;
}
```

オブジェクトは、１つ以上の種類のメッセージを受け取ることができます。もちろん、そのオブジェクトには各メッセージに対応したメソッドが用意されていなければなりません。下の例は、３つのメソッドを記述したものです（正確には書いていません）。このオブジェクトは、sleepShortTimeと、playPianoと、eatFoodを受け取ることができます。

```
sleepShortTime ( ) {  ...go to bed;  ..... zzzz ....wake up; }

playPiano ( ) {  ...open the cover; touch the key elegantly..    }

eatFood ( ) {  .....take a carrot;  hum hum ..... }
```

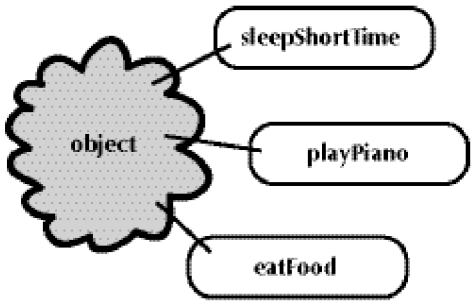

図3-5 ３つのメソッドを持つオブジェクト

結局、コンピュータのユーザは、受取り手であるオブジェクトに用意されているメソッドに対してメッセージを送って、何かをコンピュータに実行させることになります。これが、オブジェクト指向の基本的な考え方です。他力依存型と考えてもよいでしょう。ただし、プログラマはメッセージを受け取った後の処理をプログラムとして記述しておかなければなりません。メッセージを送ったり、受け取ったりすることをメッセージの受渡し（messagepassing）と呼ばれています。これは、実際にはオブジェクトに記述されたメソッドを実行することを意味していますので、メソッド呼出し（method call）とも呼ばれています。メソッド呼出しも１つの文になっています。Javaでは、メソッドを呼び出すときも、メソッドであることを示すために丸括弧（）をつけています。

```
rabbitMelo.sleepShort Time( ) ;
rabbitMelo.eatFood( ) ;
```

上の例は、*rabbitMelo*と名づけたオブジェクトに対して、sleepShortTme（ちょっと寝なさい）というメッセージとeatFood（食べ物を食べなさい）というメッセージを送っている、すなわち定義されたメソッドを呼び出している記述です。２つのメソッド呼び出しの文から構成されています。オブジェクトの名前とメソッドの名前を分けるためのドット（.）に注意してください。これは、日本語で言えば「の」に該当します。上の方の記述は、「rabbitMeloのsleepShortTimeメソッドを呼び出しなさい」という意味になっています。

呼出し側のオブジェクトは、他のオブジェクトに対してメソッドを呼び出した後は、そのメソッドの実行が終わるまで待っています。そして、メソッドの処理が終了した後に、処理を再開します。

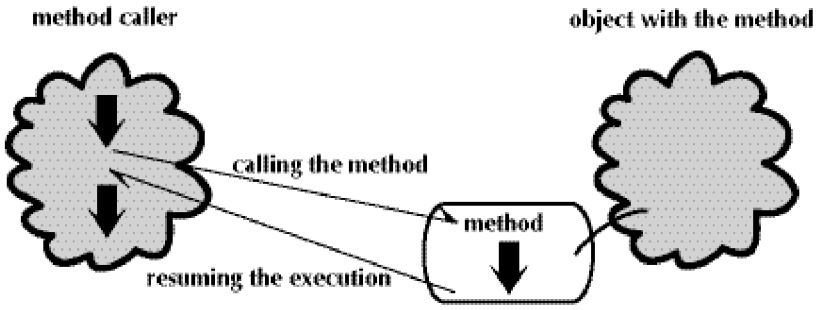

図3-6 メソッドの実行が終わるまで待ち、実行を再開する

ここまでは、一つのメソッド呼出しだけを説明しましたが、実際には更に、呼び出されたメソッドの中で別のオブジェクトに対して要求することもあるでしょう。つまり、自分で処理しないで、さらに他人に一部をお願いするということです。

また、メソッド呼出しのときは、メッセージにいろいろなデータをつけ加えることができます。これらのデータ値のことをパラメータ（Parameter）あるいは引数（Argument）と呼びます。例えば、次のようなメッセージを考えてみましょう。

```
Mammy,  pass  me  4 cheeses ;
My rabbit,  drink a cup of carrot  juice ;
```

ここでは、受取り手（それぞれ母親とうさぎ）に対して、４個のチーズ、一杯のにんじんジュースという具合にパラメータを指定しています。メッセージの受取り側のメソッドも、送られたパラメータによって動作を変えることができます。Javaでは、パラメータは呼出し側も受け取り側のメソッドも丸括弧の中にその旨を記述します。

```
//  呼出し側
Mammy.passMeCheeses( 4  );      //  この4がパラメータです。

//  受け取るメソッドの側
passMeCheeses(  number  )       {       //  numberという名前でパラメータを受け取ります
        .... go to dining ;
        ......
}
```

３－４－２．Javaでのメソッド呼出し

Java言語では、実際にメソッド呼出しをどう記述するのかをもう少し正確に紹介しましょう。メソッド呼出しの場合には、つぎのようにメッセージの受取り手のオブジェクトと、メッセージ（メソッドの名前）とパラメータを記述します。

▼メソッド呼出し文の書式

```
オブジェクトの名前．メソッドの名前（  パラメータ  ）；
```

パラメータは、カンマ（, ）で区切っていくつも書くことができます。例えば、前章のプログラムで出てきた次のような２つの記述を解釈してみましょう。

```
System.out.println( "今日はマグロ" );
        →オブジェクトの名前は、        System.out
        →メソッドの名前は、            println
        →パラメータは、                "今日はマグロ"

g.drawString( "春眠暁を覚えず", 50, 25 );
        →オブジェクトの名前は、        g
        →メソッドの名前は、            drawString
        →パラメータは、                "春眠暁を覚えず", 50, 25  の３つ
```

なお、ProjectBuilderが最初に表示するアプレットのプログラムには、setLayoutという名前のメソッドがあります。これには、呼び出す相手側のメソッドが指定されていません。これは、このアプレットのクラスは、上位クラスのAppletクラスを継承する形で定義されているからです。Appletクラスの更に上位クラスであるComponentクラスの中に、setLayoutメソッドが定義されています。つまり、上位のクラスから継承して、ここでプログラムとして提示されているアプレット自身に、そのメソッドが備わっていると解釈されるからです。

setLayout(**null**);

→オブジェクトの指定はなし	上位クラスから継承し、自分自身が持つメソッドなので
→メソッドの名前は、	setLayout
→パラメータは、	**null** （「何も指定せず」という意味）

３－４－３．Javaでの受け取り側のメソッドの定義

一方、呼び出されるメソッドをJava言語で定義するのはどのようにするのか紹介します。メソッドの定義はつぎのような書式に則って行われます。

▼メソッドの定義の書式

```
メソッドの型　　メソッドの名前（パラメータの型　受け取ったパラメータ）　　{

                メソッドの中ですべきこと
}
```

メソッドの型については、後の章で詳しく説明します。メソッドを定義するときはそのようなものを書かなければいけないということだけ覚えておいてください。なお、次の章では基本的な型を紹介します。また、受け取ったパラメータも、メソッドを呼出すときと同じように、カンマ（, ）で区切って複数書くことができます。なお、メソッドは単体では存在するのではなく、なんらかのオブジェクトの中にメソッドの定義するのがJavaの基本的なプログラムとなっています。

前章のプログラムで出てきました２つのメソッドの定義を解釈してみましょう。

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

→メソッドの型は、	**public static void**
→メソッドの名前は、	main
→パラメータの型は、	String [ ]
→受け取ったパラメータは、	*args*

public void paint(Graphics *g*) {
　　……
}

→メソッドの型は、	**public void**
→メソッドの名前は、	paint
→パラメータの型は、	Graphics
→受け取ったパラメータは、	*g*

メッセージの受渡し、つまりメソッドの呼出しには、呼び出す側（送り手）と呼び出される側（受取り手）の両方での記述が必要ということがわかっていただけましたでしょうか。なお、パラメータについては、呼び出す側で指定したパラメータには、実際の値が指定されていますので、これを実パラメータ（Actual Parameter）と呼ぶことがあります。受け取った側、すなわちメソッドの定義の際には、仮に名前を付けてそのパラメータを受け取っておいて、メソッドの中の処理ではその名前でパラメータを参照するということから、仮パラメータ（FormalParameter）と呼ぶことがあります。上の２つの定義の中では、*args*と*g*が仮パラメータになっています。

３−5. アプリケーションの実行

さて、長々とオブジェクト指向モデルについて説明してきましたが、これは、すべて前章で出てきましたプログラムの解釈を行なうためのものだったのです。まずは、アプリケーションのプログラムから考えていきましょう。Java言語で書かれたソーステキストファイルには、実は複数のクラスを定義することができます。しかしながら、アプリケーションの場合は、そのうち一つのクラスは、ファイル名と同じ名前の代表クラスでなければならないのです。ですから、ファイル名とクラス名を同じにしなければならないのです。

FirstApplication	クラス名
FirstApplication.java	ファイル名

この代表クラスは、パブリッククラス（public class）と呼ばれています。正確に言えば、パブリッククラスは、外部から利用することができるクラスなのです。パブリッククラスのときだけは、クラスの定義では**class**ではなくて、**public class**と記述する必要があります。

Javaの場合、通常アプリケーションでは、実行されるとファイルの中で定義されたパブリッククラスのmainと呼ばれる名前のメソッドを呼び出すことが決まっています。

さて、だいぶアプリケーションの規則を述べさせていただきました。ここで、最初に出てきたプログラムを、もう一度解読してみましょう。//の後に続く文章は、前に紹介したようにコメントと呼ばれています。プログラムの実行には、特に関係ありません。

```
public class FirstApplication {                    //パブリッククラスFirstApplicationの定義
    public static void main( String [ ] args ) {  // メソッドmainの定義
            System.out.println("Hello, World!");    // println というメソッドの呼出し
    }
}
```

３−6. アプレットの実行

３−６−１. アプレットに用意するメソッド

次にアプレットのプログラムを解釈してみましょう。アプレットの場合には、mainメソッドだけではなくて、さまざな場合に呼び出されるメソッドを定義しなければならないのです。つぎのような５つのメソッドを場合に応じて定義することができます。しかし、取り敢えずpaintメソッドだけを定義しておけば、何かしら描画はしてくれるという訳です。initメソッドとdestroyメソッド以外は、場合によっては何回も呼び出される可能性があります。

init( )	Webブラウザに呼び出されたとき
paint(Graphics *g*)	描画時
start( )	意識的にアプレットを開始させるとき
stop( )	意識的にアプレットを終了させるとき
destroy( )	Webブラウザ上から破棄するとき

paintメソッド以外ののメソッドの定義の仕方は、徐々に紹介していきたいと思います。また、アプレットの場合には、いろいろなクラスを利用するような形でプログラムを組んで行かないといけないのです。そのために、Javaの環境で用意されているクラスを利用するための指定をしています。この指定の仕方は次のような書式で行なわれます。

▼他のクラスを利用するための書式

```
import　パッケージ名．クラス名；
```

パッケージ（Package）とは、クラスが複数詰められているファイルのことです。このようなものを、今までの言語では、ライブラリ（Library）と呼んでいました。オブジェクト指向モデルでは、一般にクラスライブラリ（Class　Library）と呼んでいます。これは、いろいろなクラスを参照できる図書館のようなものだと思ってください。例えば、次の２つの記述は、両方ともjava.awtという名前のパッケージを利用することを記述しています。

```
import java.awt.* ;          // java.awtパッケージのすべてのクラスを利用する
import java.awt.Color;       // java.awtパッケージのColorクラスを利用する
```

コンパイラは、このような記述があったら、予めどこかに保管されているクラスライブラリから、指定されたクラスを探してきてくれます。

３－６－２．HTMLからのアプレットの実行呼出し

アプレットを実際にHTML文書の中から指定して実行してやる必要があることは前の章でも触れた通りです。ここでは、そのための記述の仕方を説明いたします。アプレットを実行させるためのタグは、appletタグと呼ばれていて、一番簡単に記述する場合は、次のように記述します。

▼appletタグの記述の仕方

```
<applet　code="実行するクラスの名前"　width="描画領域の幅"　height="高さ">
</applet>
```

例えば、次のように記述します。AppletViewerでみるときは、appletタグの部分しかみていませんので、他の部分を記述しても無視されます。この例は、幅300、高さ200のサイズの描画領域を設定して、そこにGoodWebという名前のクラスを実行させるように指定しています。なお、HTMLの版によっては、値の両側についているダブルクォーテーション記号（"）を省略しても構いません。

```
<html>
<body>
<h1>私のページです</h1>
<applet　code="GoodWeb"　width="300"　height="200"></applet>
</body>
</html>
```

また次のように、codebaseタグを使って、クラスファイルの入っているフォルダを指定したり、archiveタグを使って、Jar・Zip形式のアーカイブファイルを指定することもできます。

```
<applet code="GoodWeb"　codebase="build"　archive="GoodWeb.jar"
        width ="300"　height="200" > </applet>
```

ところが、HTMLの規格の変遷に伴い、タグの記述の仕方も増えてきました。

HTML 4.0版	appletタグの替わりにobjectタグを用いる
Netscape HTML 3.2版	appletタグの替わりにembedタグを用いる

たとえば、HTML 4.0版の場合は、次のように記述します。

```
<object   width="300" height="200" >
          <param   name = "code"   value = "GoodWeb.class">
          <param   name = "object"    value = "GoodWeb" >
          <param   name = "archive"    value = "GoodWeb.jar">
</object>
```

NetscapeのHTML　3.2版のembedタグの記述の方法は、appletタグのappletの部分が、embedに替わるだけで、後は記述の仕方は、appletタグの場合と同じです。

３－６－３．Webブラウザ上に作られる描画領域の座標系

アプレットは、WebのHTML文書を表示する際に、その一部を描画領域として指定され、その中に何かを描画していきます。この描画領域の座標系は、今までのものと少し異なるので注意してください。描画領域の座標は、左上が零点になります。x座標に関しては、右方向で通常のものと同じなのですが、y座標に関しては、下向きにプラスになることに注意点してください。以下の図は、幅（width）が300、高さ（height）が200のときのおおまかな座標イメージです。

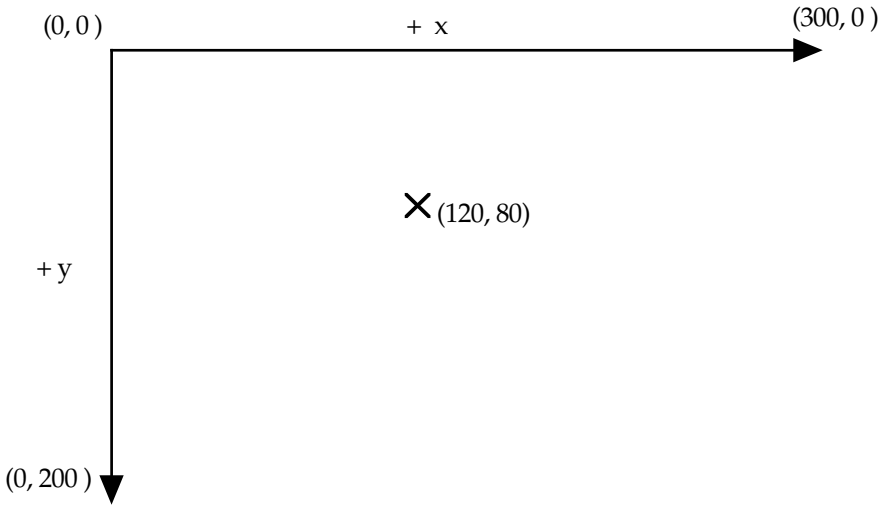

図3-10 描画領域の座標系

座標の単位は、ドット（dot）あるいは色がついたものはピクセル（pixel）と呼ばれています。整数の値で指定します。この単位は、１つの点のことを示しています。300ドットと言えば、300個の点から構成される距離を示しています。

３－６－４．描画のためのメソッド

アプレットでは、paintメソッドで描画領域を仮パラメータとして受け取っています。先のプログラムでは、受け取った描画領域を*g*という名前で受け取っています。描画領域は、グラフィックスクラスのオブジェクトと呼ばれていますが、このオブジェクトには、描画のためのさまざまなメソッドが用意されています。詳細は、後の章に譲ることにしまして、ここでは、今まで出てきたメソッドを併せて、２つのメソッドの呼出し方法を紹介しましょう。

▼drawStringとsetColorの呼出しの書式
drawString("書きたいこと", x座標 , y座標);
　　→書きたいことを描画領域上に表示してくれる。

setColor(カラー指定) ;
→それ以降の描画は、指定したカラーで表示される。

使用例として、次のような記述をしてみます。赤い色で、Hello, Apple!と表示させる例です。

```
g.setColor( Color.red );                    // これ以降はすべて赤色で描画されます。
g.drawString( "Hello, Apple!", 100, 95 );   // x座標100、y座標95の位置に表示されます。
```

上の例では、x座標100、y座標95の位置に書きたいことが表示されるのですが、これは、正確には次のような座標になっています。つまり、100, 95というのは、最初の文字の左下の座標を示しています。

図3-11 文字列の描画と指定座標

また、この例のようにカラー指定のところには、色名が使えるのですが、色名として、次のような13色が定義されています。

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

ここまでわかったところで、最初のアプレットプログラムを解釈してみましょう。ProjectBuilderがいきなりいろいろな要素が入ったプログラムをテンプレート（雛形）として作ってくれますので、解釈が大変です。まだ、説明し切れていない部分もありますが、次の章から、詳細に中身や記述の仕方を説明していきます。

```
import java.awt.* ;                         // java.awtパッケージのすべてクラスを利用する
import java.applet.* ;                      // java.appletパッケージのすべてのクラスを利用する

public class FirstApplet extends Applet {          // Appletクラスを継承したFirstAppletクラスの定義
        static final String message = "Hello, World"; // messageフィールドの定義
        private Font font =                                 // fontフィールドの定義
                new Font( "serif", Font.ITALIC +            // Fontオブジェクトの生成
                        Font.BOLD, 32 );

        public void init( ) {                               // initメソッドの定義
                setLayout( null );                          // setLayoutメソッドの呼出し
        }
        public void paint( Graphics g ) {                   // paintメソッドの定義
                g.setColor( Color.blue );                   // setColorメソッドの呼出し
                g.setFont( font );                          // setFontメソッドの呼出し
                g.drawString( message, 40, 80 );            // drawStringメソッドの呼出し
        }
}
```

いくつか説明しましたので、setColor()で設定しているカラーや、newFont()で指定しているフォントの大きさ、あるいはdrawString()で表示の位置などを変えて、もう一度実行してみて下さい。このプログラムは、まだ説明していない、いろいろな要素を含んでいますので、それらの部分を外してしまうと次のようになります。ここまでの説明で、以下のプログラムがわかっていれば良いでしょう。

```
import  java.awt.* ;                        // java.awtパッケージのすべてクラスを利用する
import  java.applet.* ;                     // java.appletパッケージのすべてのクラスを利用する

public class  FirstApplet  extends  Applet  {
                                            //Appletクラスを継承したFirstAppletクラスの定義
        public  void  paint( Graphics  g ) {                // paintメソッドの定義
                g.setColor( Color.blue );                   // setColorメソッドの呼出し
                g.drawString( "Hello, World", 40, 80 );     // drawStringメソッドの呼出し
        }
}
```

３－６－５．Webブラウザのステータスウィンドウに文字列を表示する

アプレットが、Webブラウザやアプレットビューワに表示されるときには、ステータスウィンドウ（あるいはステータスバー）に文字列を表示することができます。その場合は次のようなメソッドが用意されています。

```
showStatus( "書きたいこと"  ) ;
        →書きたいことをステータスウィンドウに表示してくれる。
```

ただし、ステータスウィンドウは短い一行であったりすることもありますので、あまり長いメッセージを表示したり、何行も表示することは避けた方がよいと思われます。次のアプレットプログラムは、アプレットの描画領域にも、ステータスウィンドウにもメッセージを表示します。

```
import  java.awt.* ;
import  java.applet.* ;

public  class  StatusApplet  extends  Applet  {
        public  void  paint(  Graphics  gc ) {
                gc.drawString( "Status Example",  100, 50 );
                showStatus( "I  am  fine,  thank you." );
        }
}
```

３－7．エラーの対処

３－７－１．エラー表示の例

コンパイラは、構文エラーがあれば、該当箇所を表示して、英語（あるいは日本語）でエラーの内容を表示してくれます。基本的には、ファイル名、行番号、何が悪かったか、該当箇所、停止するまでにいくつエラーがあったかなどを表示してくれます。SUNのJDKの英語でのエラー表示の例をいくつか見てみましょう。これらのメッセージは、ProjectBuilderで、ビルドの途中画面を、更にウィンドウを分割して、表示してくれますので、実際に見ることができます。

```
Hello.java:4: Identifier expected.  （識別子がありません）
      public statec void main( String arg[] ) {
                  ^
```

これは、４行めに問題があって、staticとするところをstatecと入力してしまったものです。コンパイラが ^ の位置にある単語がおかしいことを示しています。

```
Hello.java:5: ';' expected.　（セミコロンが必要です）
                System.out.println( "Tako" )
                                    ^
```

５行目がおかしいようです。最後のセミコロン（;）が抜けているみたいです。

```
Hello.java:6: '}' expected.（} が必要です）
}
 ^
```

最後の}を書き忘れたようです。波括弧（Braces）は、開いたら（ {を書いたら）閉じる（} を書く）ということを忘れないでください。

```
Hello.java:5: Method println(java.lang.String) not found in class java.io.PrintStream.
          （java.io.PrintStreamクラスには、println というメソッドはありませんよ）
            System.out.println( "Tako" );
                               ^
1 error
```

printlnとかくところをprintlnと書いてしまったようです。大文字のアイ（I）と、小文字のエル（l）、そして数字の１は、特に間違いやすいので綴りミスに注意してください。

３－７－２．間違いやすいが発見しにくい構文エラーの例

よく起こしてしまう間違いで、特に自分自身では発見しにくいものの例をいくつか挙げてみます。

★ダブルクォーテーション（”）を閉じ忘れた場合です。後半の”が抜けています。

例：　System.out.println("This is a sample.);

★最後のセミコロン（ ; ）を付け忘れた場合です。

例：　System.out.println("This is a sample.")

★記号が日本語で入力されている場合です。

記号は、半角の英字で入力しなければなりません。間違いやすい記号として、次の幾つがあります。

”　　文字列が終了していないと見なされてしまいます。
□（スペース）　「\64は不正な文字です」と表示されていたら、どこかに全角の空白が入っています。
（ { 　} ）；　括弧などが全角の記号になっているときは見分けにくいですね。

★その他の間違いやすいが見つけにくい例

・ピリオド（. ）を入力していない
・Systemとout、およびoutとprintlnの間にピリオドを入れた後に空白をあけてしまった。
・括弧を閉じ忘れた。逆に余計に閉じてしまった。括弧は] や) あるいは、} 等です。
・名前が違っている。println→println（小文字のエル）
・大文字にしていない。system→System

３－８．課題

課題3-1
二重引用符 "（Double quotation mark）で囲まれた文字列はHello, World!という英語でしたが、これを日本語の文字列にしてみて、うまく表示できるかどうか確かめなさい。アプリケーションでもアプレットでも両方やってみなさい。アプリケーションの方では、JJEditを使いなさい。

課題3-2
アプリケーションプログラムで、System.out.printlnやSystem.out.printを複数使って、つぎのような文章を表示するようなプログラムを書いてみなさい。ファイル名は、JITMessage.javaとしましょう（よってクラス名もJITMessageとしてください）。

> The java command executes Java bytecodes created by the Java compiler, javac. The JIT (just in time) compiler compiles these bytecodes into machine instructions by default in the Solaris Java VM when using java.