

Chapter 3.

プログラムの記述とオブジェクト指向について

3.1. プログラムの記述の仕方

■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、次のような記述があったとしましょう。

```
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。命令を使っている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。

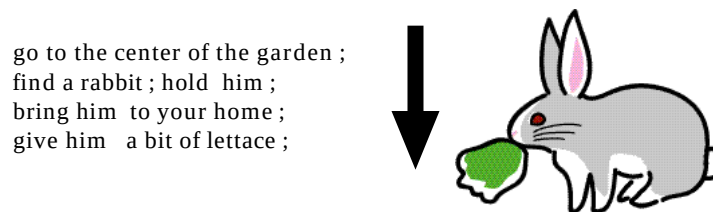


図 3-1 文の例

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（Imperative Programming Language）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。

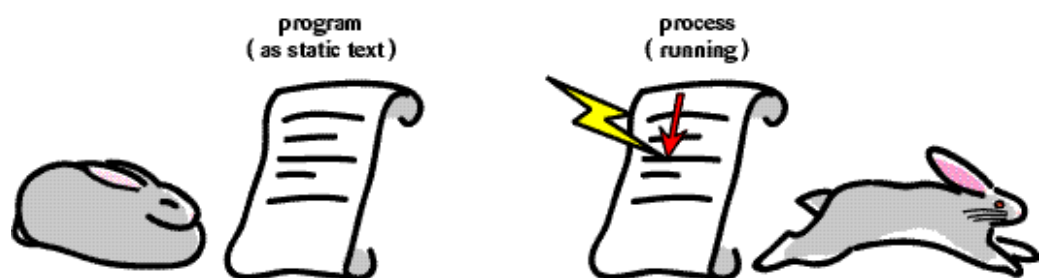


図 3-2 プロセスとプログラム

■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 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；  
    ：  
}
```



■Java言語でのコメントの付け方

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、//以降から行末に掛けられている文字は、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  
    }  
}
```

■改行や空白について

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。

3.2. オブジェクト指向について

■オブジェクトとメッセージの受け渡し

オブジェクト指向モデル（Object Oriented Model）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つ走り出すだけだったのですが、オブジェクトは、プロセスをさらに分解したものとして考えることができます。つまり、一つのプロセスは、複数のオブジェクトから構成（Composed）されるのです。

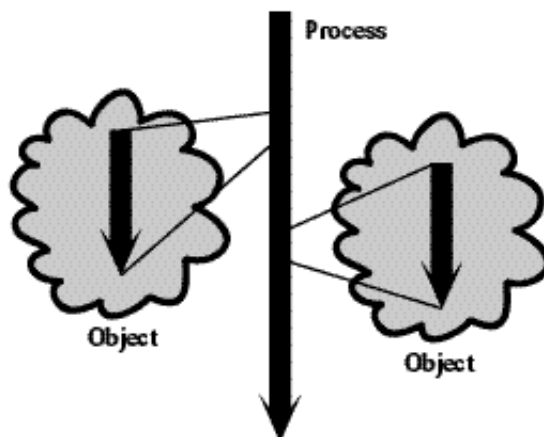


図3-3 プロセスとオブジェクト

しかし、オブジェクト自身はかなり受動的な部品なのです。他のオブジェクトから、何かを要求されるまでは動き出しません（注1）。要求されて初めて、オブジェクトの中に格納されているプログラムを実行し始めます。そのため、いくつかのオブジェクトを全体として観測してみると、オブジェクトは、始終互いに要求を出すために通信しあっているように見えます（ちょうど携帯電話を持って、いつでも話しながら仕事をしている人のように）。なぜ、通信することによって、処理が進んで行くのか考えてみましょう。

まず、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。

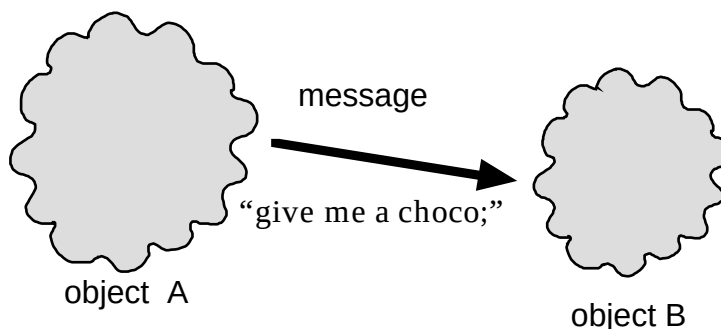


図3-4 オブジェクト間でのメッセージの送受

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。

```
giveMe( a choco ) {                               -- when object receives "giveMe a choco" message
    take a choco from the refrigerator;
    pass a choco 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 ..... }
```

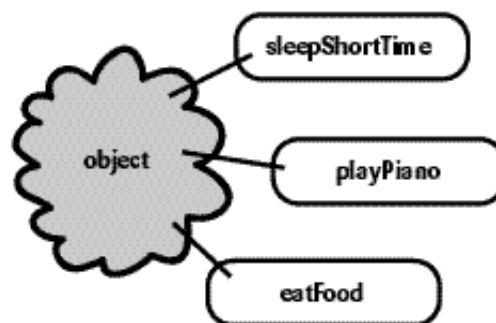


図3-5 3つのメソッドを持つオブジェクト

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、プログラム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（message passing）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。

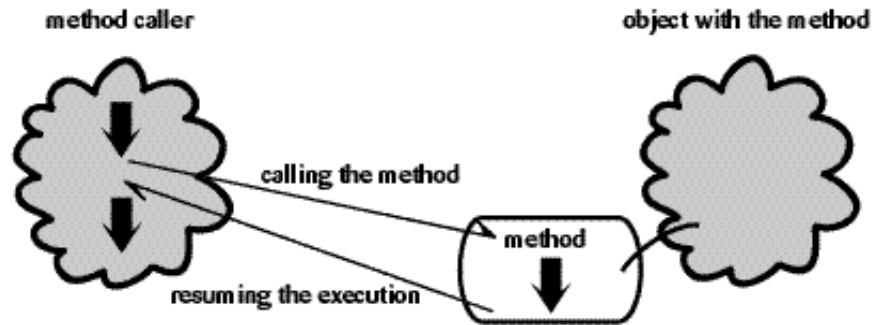


図3-6 メソッドの実行が終わるまで待ち、実行を再開する

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事です。例えば、次のメソッドの記述は、チケットが欲しいという依頼者からのメッセージに対するメソッドの記述の例です。ここでは、お金を自分で用意して、更に別のオブジェクトである Butler（執事という意味です）に対して、お金を渡して店に行ってチケットを買ってくるように要求しています。オブジェクトButlerは、店に行って、更に別のオブジェクトに対して、ticket を売ってくれるように要求するかもしれません。

```
get Ticket( ) {  
    makeSomeMoney( );  
    Butler.goToStore( );  
    Butler.buyTicket( );  
    return theAcquiredTicket;  
}
```

このように、呼び出されたメソッドの中で、更に別のメソッド呼出しが発生することになるのです。オブジェクト指向モデルにおいては、このようにオブジェクトが別のオブジェクトのメソッドを呼ぶことが連鎖的に行なわれることによって、計算が進んでいきます。これが、オブジェクトが始終通信しあっている理由なのです。

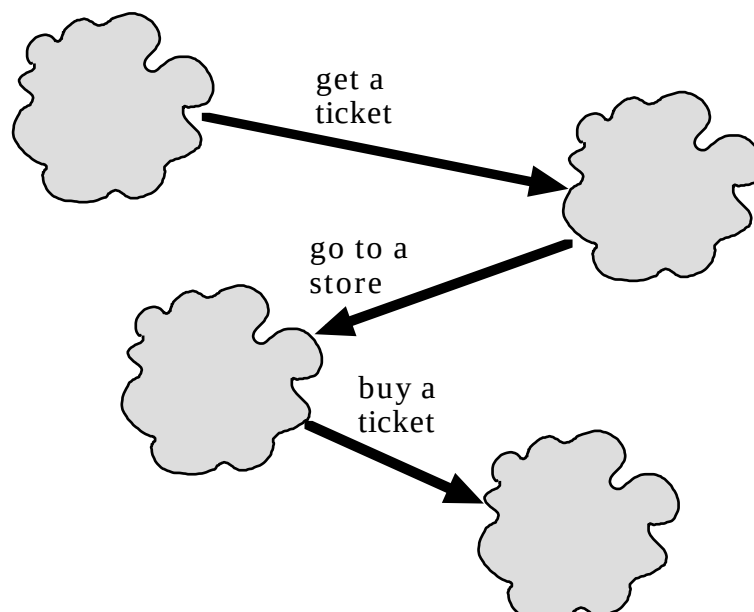


図3-7 なんとなく他人任せに見えるメッセージの受渡しの連鎖

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, give me 4 chocos ;  
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つのチョコレート、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.giveMeChocos( 4 );    // この4がパラメータです。  
  
// 受け取るメソッドの側  
giveMeChocos( number )    {    // numberという名前でパラメータを受け取ります  
    .... go to dining ;  
    .....  
}
```

注1 独自に走り出すオブジェクトのことをエージェント（Agent）と呼ぶこともあります。

■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"  
  
gc.drawString( "春眠暁を覚えず", 50, 25 );  
→オブジェクトの名前は、    gc  
→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
```

■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 gc ){  
    .....  
}
```

→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、	gc

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（Formal Parameter）と呼ぶことがあります。上の2つの定義の中では、argsとgcが仮パラメータになっています。

■クラスとオブジェクトについて

オブジェクト指向モデルでは、同種のオブジェクトは、クラス（Class）として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特性（Property）が記述されています。特性の中には、次のような内容が記述されている。

- ・オブジェクトが実行できるメソッドの定義
- ・オブジェクトが保持する属性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。

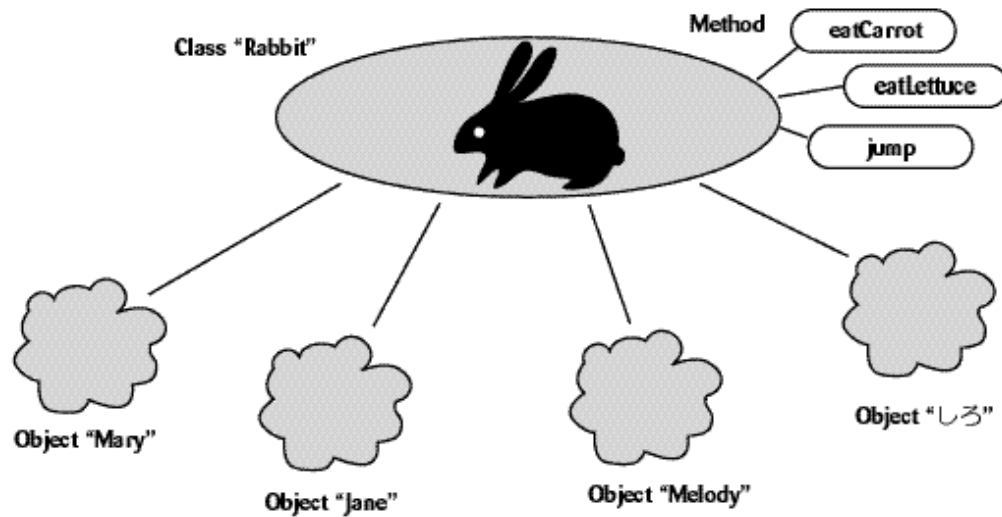
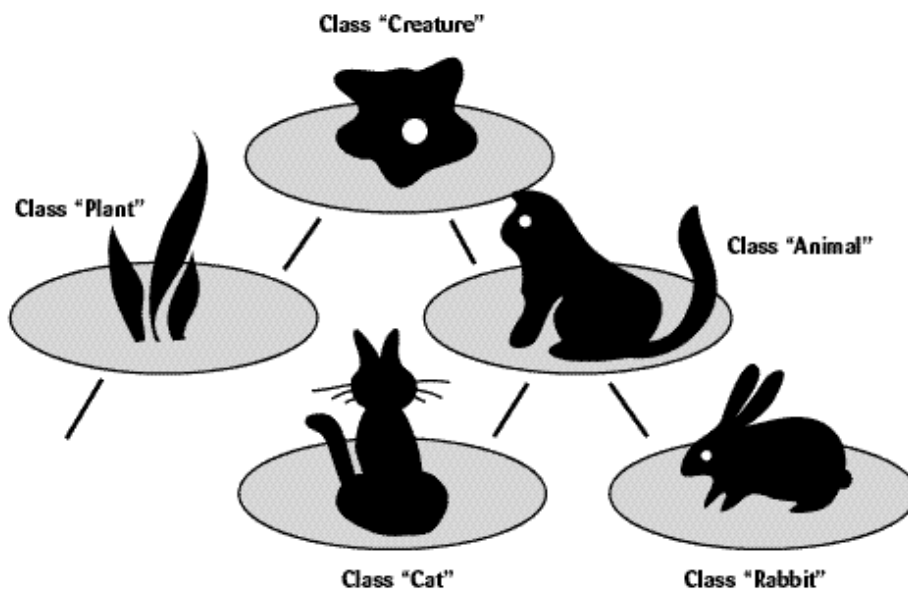


図3-8 クラスとオブジェクト

■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。



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。

■Javaでのクラス定義

Java言語で、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 {

    メソッドの定義やフィールドの定義など

}
```

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。以下のクラス `Rabbit` は、`eat` という名前の一つのメソッドを持っています。....の部分は、何かそこに書いてあるという意味です。

```
class Rabbit {
    public void eat( Food f ) { ..... }
}
```

次に、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。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 {

    メソッドの定義やフィールドの定義など

}
```

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

このようにして、一つのオブジェクトを記述するのにさまざまな工夫があるのです。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. Javaでのオブジェクト指向プログラミング

■アプリケーションとクラスについて

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。

Hello	クラス名
Hello.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 Hello {                                // パブリッククラス Hello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
    }
}
```

最初のプログラムの意味がわかりましたか？

■アプレットの場合に呼び出されるメソッド

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、さまざま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つのメソッドの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 gc)	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。

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。

```
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 HelloWeb extends Applet {           // Appletクラスを継承したHelloWebクラスの定義
    public void paint( Graphics gc ) {           // paintメソッドの定義
        gc.drawString( "Hello, Window!", 50, 25 ); // drawStringメソッドの呼出し
    }
}
```

■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、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けませんので、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。

```
<html>
<body>
<h1>私のページです</h1>
```

```
<applet code=GoodWeb width=300 height=200></applet>
</body>
</html>
```

■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。

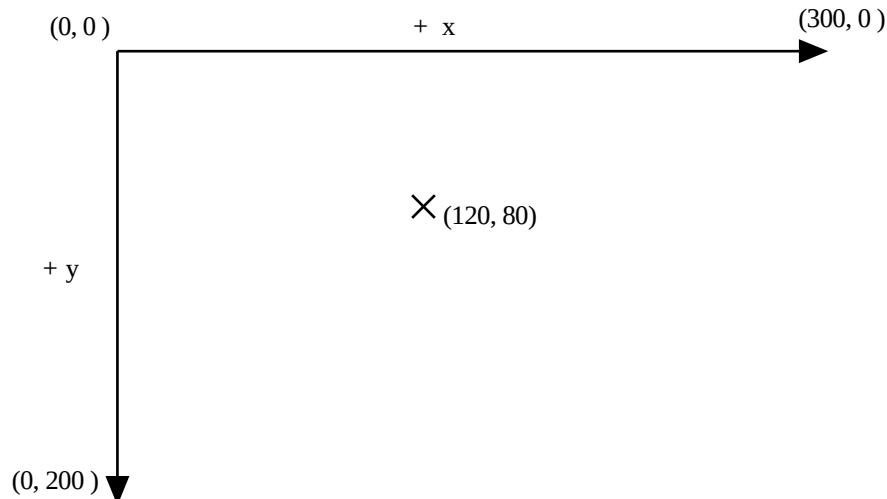


図3-10 描画領域の座標系

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。

■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gc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も併せて、2つのメソッドを紹介しましょう。

```
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。

```
gc.setColor( Color.red );           // これ以降はすべて赤色で描画されます。
gc.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	

■色を配色したアプレット

配色を施して文字を表示させるようなアプレットを作ってみましょう。やり方は、前章で示した通りです。HTMLファイルの方は、前に作ったTester.htmlをそのまま流用しましょう。ただし、Tester.htmlの内容は、次のように修正してください。

```
<applet code=GoodWeb width=300 height=200></applet>
```

プログラムは、次の通りに記述してみましょう。ファイル名は、クラス名と同じにしなければいけませんから、GoodWeb.javaとします。プログラムの内容は、もうお解りですね。

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

public class GoodWeb extends Applet {
    public void paint( Graphics gc ) {
        gc.drawString( "I am a black Message.", 50, 25 );
        gc.setColor( Color.red );
        gc.drawString( "Hello, Apple!", 100, 80 );
        gc.setColor( Color.orange );
        gc.drawString( "Hello, orange.", 200, 150 );
    }
}
```

■アプレットとWebページ

アプレットは、Netscape NavigatorやInternet Explorerでも見ることができます。Tester.htmlを指定して実際に動かかどうか確かめてみてください。また、前章の課題にありました通りに、ホームページを持っている人は、自分のホームページ上でアプレットを動かすことも可能です。ここでは、NCSA のWebサーバーやApacheという名前のWebサーバーで管理されているUnixオペレーティングシステム（SolarisやLinuxなど）で、自分のホームページでアプレットを動かす方法について、簡単に解説します。その他の場合は、各プロバイダや学校組織あるいは企業で方法は異なると思いますので、各自で調べてみて下さい。だいたいは、ファイ

ルをWebサーバーが稼働しているマシンに転送するような形になります。

例えば、Solarisではユーザのホームディレクトリの下にpublic_htmlという名前（この名前は異なるかも知れませんが）のディレクトリが用意されていて、そこにHTMLのファイルと、クラスファイルを転送すれば良いという場合が多いと思います。public_htmlのディレクトリの下にファイルをコピーするには、次のように行ないます。この例では、Tester.htmlというファイルと、GoodWeb.classというファイルを転送しています。

```
% cp Tester.html ~/public_html
% cp GoodWeb.class ~/public_html
```

cpは、コピーするコマンドです。移動したいときは、mvを用います。%は、システムが表示してくれているものです。この操作をNetscape Navigatorを立ち挙げて、つぎのようにURLを打ち込んでみて下さい。

```
http://サーバマシン名/~自分のログイン名/Tester.html
```

自分のログイン名の部分は、各組織で用意されているユーザ名です。ところで、なぜソーステキスト（.javaで終わるファイル）は、転送しなくていいのでしょうか？わからない方は、第1章をもう一度読み直してみてください。

■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時には、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 StatusWeb extends Applet {
    public void paint( Graphics gc ) {
        gc.drawString( "Status Example", 100, 50 );
        showStatus( "I am fine, thank you." );
    }
}
```

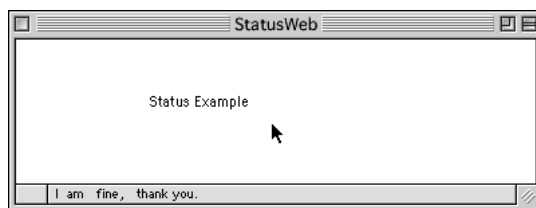


図3-12 Apple Applet Runnerで表示した実行例

3.4. まとめと課題

■まとめ

★命令型プログラム

- ・一つの命令を文と呼ぶ
- ・命令文が順次実行されていく

★オブジェクトとプロセス

- ・実行の単位プロセスがさらに分解されたのがオブジェクト
- ・オブジェクトは命令を受け取る対象

★メッセージの受渡し（メソッドの呼出し）

- ・送り手がメッセージを出す
- ・受取り手がメッセージに対応したメソッドを持っている
- ・メッセージにはパラメータを添付することができる

★クラス

- ・同種のオブジェクトはクラスという形で定義される
- ・クラスには特性（メソッドの定義とフィールドの定義）がなされている
- ・そのクラスに属するオブジェクトをインスタンスと呼ぶ
- ・クラス間には継承関係がある

★Javaの記述の仕方

- ・メソッドの呼出しの仕方
- ・メソッドの定義の仕方
- ・クラスの定義の仕方
- ・継承関係を利用したクラスの定義の仕方

★アプリケーションとアプレット

- ・アプリケーションでの代表クラスがpublic class
- ・アプレットでは、場合に応じたいくつかのメソッドが定義できる
- ・予め用意されたクラスは、パッケージにしてまとめられている
- ・描画領域の指定の仕方、座標の取り方
- ・色などを使った描画
- ・自分のホームページに置いてみる

■課題

3-1

8つの色を使って、次のように表示させるアプレットのプログラムを作りなさい。どの行にどの色を用いても構いませんが、必ず異なる8色を用いなさい。最後の行の始まる位置にも注意しましょう。行間、開始の座標は、適宜各自で定めましょう。400x300のサイズの描画領域中に表示できるように指定しなさい。

*Somewhere, over the rainbow,
way up high,
There's a land that I heard of once in a lullaby.
Somewhere, over the rainbow,
skies are blue,
And the dreams that you dare
to dream really do come true.*

Lyrics By: E.Y. Harburg

3-2

生物に関する正確なクラス階層を設計してみなさい。たとえば、脊椎動物とか哺乳類などのクラスを考慮してみなさい。それを図に書いてみましょう。クラス名として英語の単語を調べてみなさい。