Chapter 13. アプレットとイベント駆動プログラミング

１３－１．アプレット

アプレットがSafariやAppletViewerなどのWebブラウザの中で、実際に実行されるときのことをもう一度考えてみましょう。

まず、アプレットのプログラムがHTMLファイルの指定によって、NetscapeなどのWebブラウザに読み込まれたときに、プログラムに書かれているクラスの定義を参照して、その定義に基づいたインスタンスが一つ作られます。そして、Webブラウザが、そのインスタンスが持つメソッドを事ある度に呼びます。

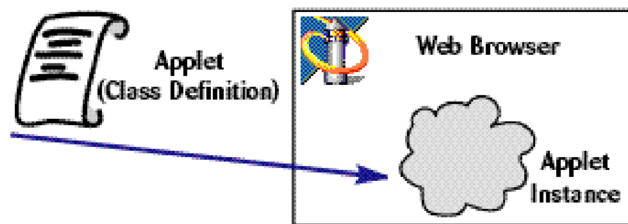

図13-1　アプレットのインスタンスと実行

Appletクラスには、次のようなメソッドが用意されていて、Webブラウザから呼び出されることになっています。これらのメソッドは、指定しなければ特別なことは何もしないようになっています。もし何かするようにしたければ、自分で同じ名前（シグネチャ）のメソッドを用意して、Appletクラスのメソッドではなくて、自分で定義したメソッドの方を呼んでもらうようにします（注１）。

メソッド名	メソッド呼出しの起こるタイミング
init	アプレットがWebブラウザに読み込まれたときに
paint	Webブラウザがアプレットに描画を要求したときに
start	Webブラウザがページ（HTMLファイル）を描画するときに
stop	ユーザが他のページを見に行ったときに
destroy	もうアプレットがいらなくなったとき
update	アプレットの内容更新を制御するときに

initメソッドは、最初に１回だけWebブラウザに呼ばれて実行されます。それに対して、paintメソッドや、startメソッドは、描画が要求されたとき、あるいはユーザが他のページを見に行ってもう一度戻ってきたときに、Webブラウザから毎回も呼び出されるという違いがあります。updateメソッドは、通常では定義することはほとんどありません。アニメーションを高速に描画したい場合に、再描画する範囲を指定するときなどに限って定義することがあります。

このように、いろいろな事象（イベント）が起こり、その事象に応じたメソッドが呼ばれるような機構のことを、イベント駆動（Event Driven）方式と呼んでいます。

今までのアプレットプログラムでは、単に描画だけを行なっていたので、描画を行なうメソッド、すなわちpaintメソッドだけを用意すれば良かったのです。しかし、必要であれば上に掲げたその他のメソッドを用意する必要があります。

（注１）名前や引数などのメソッドの仕様をシグネチャ（Signature）と呼びます。スーパークラスのメソッドと同じ名前（シグネチャ）のメソッドをサブクラスで用意して、スーパークラスの方のメソッドを呼ばせないようにすることを、メソッドの上書き（Overwriting）あるいはオーバーライディング（Overriding）と呼ぶことがあります。

１３−２．アプレットの状態を覚えておくには

１３−２−１．ローカル変数とインスタンス変数

メソッドの中で宣言した変数というのは、一般的には局所変数あるいはローカル変数（Localvariable）と呼ばれています。これは、メソッドの実行が終了してしまえば、跡形もなく消えてしまう変数です。他にもブロックごとにローカル変数を宣言できることもできます。

たとえば、以下のプログラムでは、変数*x*と*y*、それから途中で変数*i*が宣言されています。これらの変数は、paintメソッド内では有効で、そのまま利用することができます。なお、このようなローカル変数は、参照する前に必ず値を代入しておかなければいけません。

```
public class Sample extends Applet {
        public void paint( Graphics gc ) {                変数 x, y が有効な領域
                int x, y;
                ........
                int i=10 ;                                 変数 i が有効な領域
                while ( i < 100 ) {
                        .......
                        i +=10 ;
                }
        }
}
```

一方、アプレットを考えると、例えばアプレットがNetscapeなどのWebブラウザ上で表示されている間、ずっと保持していたい情報などがあります。例えば、四角形や直線などを描画する最初の始点の位置などを持っていて、状況がかわるごとに、その始点の位置などをずらしたい、などというアプレットを設計したい場合を考えてみましょう。メソッドの実行が終わると消滅してしまうようなローカル変数では、このような情報をその後も保存しておくことはできません。

このような場合のために、あるクラスのインスタンスが存在し続ける間、情報を保持することができる変数を宣言することができます。これをインスタンス変数（Instance variable）と呼んでいます。なお、オブジェクトの外部からインスタンス変数を参照するときは、Javaの用語としてフィールド（Field）という名前が用いられています。

インスタンス変数は、通常クラス定義の最初の方に書かれます。

▼インスタンス変数の宣言の場所

```
class クラス名 {
        インスタンス変数の宣言

        メソッドなどの定義
}
```

たとえば、以下のクラスでは、２つのインスタンス変数*startx*、*starty*を宣言しています。これらの変数は、アプレットが存続している間、何らかの整数が保存されていて、いつでもそれを参照することができます。

```
public class PointHolder extends Applet {
        int     startx;                 // 何かの開始点のx座標
        int     starty;                 // 何かの開始点のy座標

        public void paint( Graphics gc ) {
                        ..........
        }
}
```

インスタンス変数は、クラスの中で定義されたすべてのメソッドから参照することができます。また、インスタンス変数は、ローカル変数と異なり、参照される前に必ずしも値が代入されていなくても構いません。何も代入されていない変数が参照された場合、各型で決められている初期値が用いられます。

何も代入されない場合に、インスタンス変数に初期値として仮定される値は以下の通りです。ただしコンパイラによっては、警告を出すものもあります。なるべく変数は参照する前に値を代入しておきましょう。
フィールド変数の場合は許されるのですが（注１）、そのときは次の値が予め代入されているものとします。

整数型の変数	0
実数型の変数	0.0
論理型の変数	**false**
文字型の変数	Unicodeのヌル文字（\u0000）
文字列型の変数	**null**（何も文字列が設定されていないことを示す）
配列型の変数	**null**（配列の領域が設定されていないことを示す）

インスタンス変数に初期値が代入されている場合は、クラスからインスタンスが作られる際に、その代入が実行されると思って構いません。次の例では、PointHolderクラスのアプレットが作られたら、最初に*startx*=10と*starty*=25が実行されます。

```
public  class  PointHolder  extends  Applet  {
        int      startx = 10;
        int      starty = 25;

        public void  paint( Graphics  g  )  {
                g.drawRect( startx, starty,  20,  20  );
                startx += 10;   starty += 25;                    //  描画し直すたびに位置を変える
        }
}
```

★アプレットとインスタンス変数

アプレットの場合、インスタンス変数はアプレットの状態を保持しておくために主に用いられます。そのために、インスタンス変数をinit メソッドで初期化するのが定石になっています。通常の整数などを保持する変数の場合は、宣言時に代入することができます。オブジェクトを参照する変数の場合は、宣言時の初期値代入を使うこともありますが、initメソッドで行なう方がわかりやすいでしょう。

```
public  class  PlaceHolder  extends  Applet   {
        int                 startx = 10, starty = 100;     // 宣言値の初期値代入
        FontMetrics         metrics;

        public  void  init(  )  {
                metrics  =  getFontMetrics( );         //  initメソッドの中で代入
        }
        ........
}
```

★別ウィンドウを制御する

前の章で出てきたFrameクラスを使ってみましょう。このクラスは、アプレットとは別のウィンドウを出すときに使われます。このクラスのオブジェクトでは、以下のようなメソッドが用意されています。

new Frame("ウィンドウのタイトル")	新しくウィンドウを作ります
setSize(幅,　高さ)	ウィンドウの幅と高さを設定します
show( )	画面上にウィンドウを配置します
setVisible(論理値)	ウィンドウを表示するかどうか指定します
dispose( )	ウィンドウを破棄します

getGraphics( )　　　　　　　　　　　　　ウィンドウの描画領域を返してくれます

setVisibleは、setVisible(**true**)でウィンドウを見せますし、setVisible(**false**)でウィンドウを隠します。それでは、これらのメソッドを使って、実際に別に１つの付属のウィンドウを持つようなアプレットを定義してみましょう。インスタンス変数*window*が、このウィンドウを示すオブジェクトを参照しています。

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

public  class  FrameTester  extends  Applet {
        Frame   window;                          //  ウィンドウのオブジェクトを指す

        public  void init( ) {
                window = new Frame( "Cool Window" );            // 初期化する
                window.setSize( 200, 200 );                     // 幅・高さを指定
                window.show( );                                 // 配置する
        }
        public  void   paint( Graphics  gc  ) {
                window.getGraphics( ).drawString( "Hello", 50, 100 );  // ウィンドウ上に描画
                gc.drawString( "Hello, Again", 50, 100 );       // アプレット上にも描画
        }
        public  void  start(  ) {
                window.setVisible( true );                      // ウィンドウも見せる
                repaint( );                                     // 再描画する
        }
        public  void  stop(  ) {
                window.setVisible( true );                      // ウィンドウを隠す
        }
        public void destroy( ) {
                window.dispose( );                              // ウィンドウを破棄
        }
}
```

アプレットに必要なメソッドをすべて定義してみました。initメソッドでは、ウィンドウを作りだし、それをインスタンス変数で参照します。そして、そのウィンドウの幅と高さを設定して、画面上に配置します。paintメソッドは、getGraphicsメソッドを利用してウィンドウの描画領域を指定します。そして、そこにdrawStringで文字列を描画しています。startメソッドは、Webブラウザでアプレットが含まれるWebページに戻ってきたときに実行されるメソッドです。このときは、setVisibleメソッドを使ってウィンドウを表示させるようにします。また、repaintメソッドを使って再描画させるようにしています。このrepaintメソッドは、次のところで解説します。stopメソッドは、アプレットが含まれるWebページからユーザが離れたときに実行されます。このときは、setVisibleメソッドを使って付属のウィンドウも隠しておきます。destroyメソッドは、アプレットが破棄されるときに呼ばれるメソッドです。このときは、付属のウィンドウもdisposeメソッドを使って破棄します。

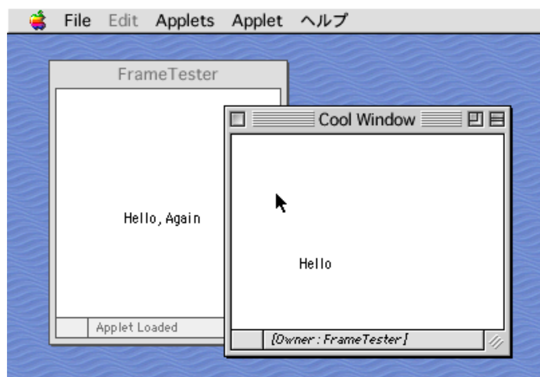

図153-2　FrameTesterの実行例

実は、このプログラムでは、付属のウィンドウだけを別の場所に移動したときに、ウィンドウ内が再描画されません。ウィンドウが別の場所に移動したことを示すイベントがアプレットの方には届かないからです。これをうまく対処するためには、スマートコンポーネントか、後で出てくるインターフェースの機能を使ってイベントの受取り手を指定するかの方法を使います。これらについては後の章を参照してください。

★repaintメソッド

repaintメソッドは、Webブラウザに再描画を要求します。Webブラウザは、アプレットが再描画するための環境を整えてから、まずはアプレット側ののpaintメソッドを実行させることになります。

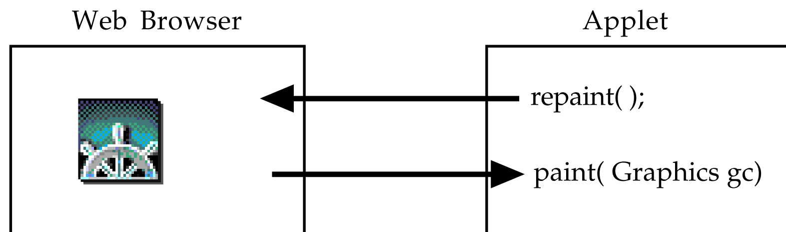

図13-3 アプレット側でrepaintメソッドを呼ぶと、Webブラウザがpaintメソッドを実行させる

アプレット側のpaintメソッドを呼び出した後、Webブラウザは、更にボタンなどの構造物を再描画しています。そのため、ボタンなどは、常に最後（最前面）に描かれることになります。

★superとthis

インスタンス変数として、Java言語には特別な２つの変数が用意されています。**super**と**this**という名前の変数です。

▼**this**はインスタンスそのものを指し示すために用いられます。例えば、アプレットのプログラムであれば、実行されているアプレット（オブジェクト）自身のことを指しています。
▼**super**は、スーパークラスのメソッドを明示的に呼び出すために用いられます。これが必要になったときに、その詳細を説明しましょう。

１３−３．インターフェース

通常、インターフェース（Interface）とは、ものごとの境界のことを意味します。例えば、マンマシン・インターフェースというのは、コンピュータと人間との接点、あるいはその仕組みや仕様のことを示しています。しかし、Javaにおけるインターフェースとは、これとはちょっと意味が異なっています。

復習から始めましょう。上位クラスから下位のクラスへ、上位のクラスで定義した性質が下位のクラスへ継承されるのがJavaでの継承（Inheritance）です。このとき、上位のクラスのことをスーパークラス（super class)、下位のクラスのことをサブクラス（sub class）と呼びます。例えば、通常プログラムとして定義しているアプレットは、Appletクラスのサブクラスとして定義しているのです。

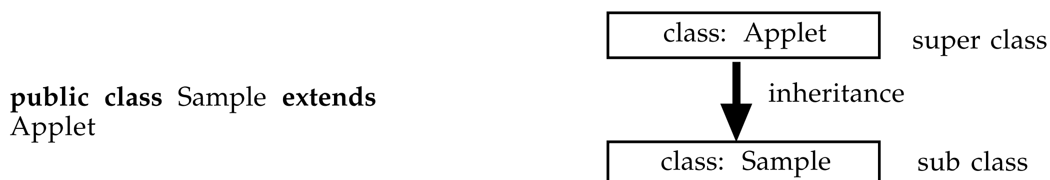

図13-4　アプレットと継承

スーパークラスのAppletに定義されている様々な性質が継承されて、自ら定義したクラスSampleも、アプレットとして振舞うことが許されます。それでは、自分で定義したクラスに、さらにいろいろな性質を継承したいとしましょう。例えば、ウィンドウを出すFrameというクラスの性質も利用したいとします。次のように書けばいいのでしょうか？

```
public class Sample extends Applet, Frame
```

実は、これは許されていないのです。Java言語では、「スーパークラスは１つしか利用できない」という規則があります。この規則は、継承のメカニズムを簡単に実現できるようにするために決められました。複数のスーパークラスを利用することができると言語的にいろいろと問題が発生するのです。

★インターフェースを含めたクラスの定義

それでは、複数のクラスの性質を利用したい場合には、どのようにすればよいのでしょうか？Java言語では、スーパークラスの替わりにインターフェースという特別なクラスが用意されています。

インターフェースクラスの例：

Runnable	スレッド（分身）を走らせるためのもの
ImageObserver	画像ファイルの読み込みを監視するためのもの
AppletContext	HTMLファイルと連携するためのもの
AudioClip	音を出すためのもの
ActionListener	ユーザの入力を受けるためのもの

このように予め定義されているインターフェースは、通常のクラスのように用いることができます。しかし、クラスとの違いが一つあります。「インターフェースは複数利用することができる」ということです。クラスを定義する際に、インタフェースを利用する場合は、次のような構文を利用します。

▼インターフェースを使うときの書式

```
class クラス名 implements インタフェース名
```

クラス名の後には、継承の指定もできます。また、利用したいインターフェースがあれば、カンマ（, ）で区切って複数指定することができます。次の例を見てください。

```
public class GreedySample extends Applet
    implements Runnable, ActionListener
```

ここで定義したクラスGreedySampleは、基本的にAppletのサブクラスですが、RunnableとActionListenerの性質も受け継いでいます。

★指定されたメソッドの実現

加えて、インターフェースでは利用する場合に実現しなければならないメソッドが指定されています。一つのメソッドの指定のことをシグネチャ（Signature）と呼ぶこともあります。実は、インターフェースには、実現されなければならないメソッドのシグネチャの集合が記述されているのです。例えば、上で利用した２つのインターフェースでは、それぞれ次のようなメソッドを定義する必要があります。

Runnableインターフェース
　→**public void** run( )というメソッドを定義する必要がある

ActionListenerインターフェース
　→**public void** actionPerformed(ActionEvent e)というメソッドを定義する必要がある

なんか、スーパークラスに比べてちょっと使うのが面倒な気もしてきましたが、なんで、こんなものが必要なのでしょうか？それは、インターフェースで定義されたメソッドさえ用意しておけば、後はシステム（例えばNetscapeなど）が自動的に、あるタイミングでそのメソッドを自動的に呼び出してくれる仕組みになっているのです。Javaでは、これを利用して様々な機能を一つのプログラムに付け加えているのです。

この「あるタイミングでシステムが自動的に実現されたメソッドを呼び出してくれる」という機構を使って、アプレットは様々なことを行っています。もちろん、この機構はAppletクラスの中に定義されているのですが、「標準で定義されているもの以外を利用する場合は、インターフェースを使う」と覚えておくといいでしょう。

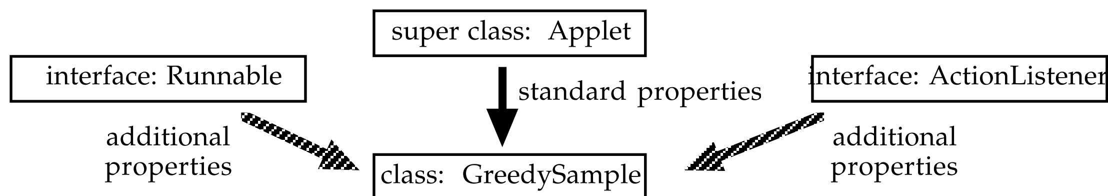

図13-5　機能が付け加わったクラスの例

Javaにおけるインターフェースは、このようにクラスに付加される仕様のことを示しています。その意味で、「インターフェース」という名前が用いられています。

１３－４．ユーザ応答するアプレット

１３－４－１．ユーザへの応答とインターフェース

さて、今まで単に描画するだけのアプレットであったのですが、ユーザからの要求に応じて、処理を変えるようなアプレットを作成してみましょう。

JDK1.1からは、ユーザの応答に答えて処理をするようなアプレットでは、標準のアプレットの仕様に加えて、ユーザからの要求を受け取るための仕様を追加しなければなりません。追加仕様は、インターフェースの項で紹介したように、ActionListenerという名前のインターフェースに定義されています。

結局、ユーザに応答するクラスの定義は、例えば次のように始まることになります。

```
public class SampleApplet extends Applet implements ActionListener {
        ......
}
```

ActionListenerインタフェースを利用するときには、actionPerformedという名前のメソッドを定義しておかなければなりません。このメソッドが、ユーザが行なった操作（イベント）に対応してWebブラウザから呼び出されます。さらに、initメソッドの中で、アプレットがWebブラウザにロードされたときに、ユーザからの操作に答えられるようなオブジェクトをインスタンス変数に用意する必要があります。

結局、ユーザに応答するようなアプレットをプログラミングするときには、以下の書式のように３つのメソッドを用意しなければなりません。また、イベントを使うためのimport文が必要になります。

▼ユーザに応答するアプレットの大まかな書式

```
import  java.awt.*;
import  java.awt.event.*;     // これが追加で必要になる
import  java.applet.*;

public class  アプレットクラス名  extends  Applet  implements  ActionListener {

        アプレットの状態を保持するインスタンス変数の宣言

        public  void  init( ) {
                アプレットがWebブラウザに読込まれたときにすること
        }

        public  void  paint( Graphics  g ) {
                アプレットに描画指令が出たときにすること
        }

        public  void  actionPerformed( ActionEvent  e ) {
                アプレット上のオブジェクトにユーザが何らかの操作を
                行なったときにすること
        }
}
```

ActionListenerインターフェースを用いる場合は、アプレット上にユーザからの操作を受けるような構造物を配置したときに限られます。このような構造物としては、次のようなものがあります。表に挙げましたように、構造物によっては、別のインターフェースを利用する場合もあります。

構造物	内容	利用するインターフェース
Button	ボタン	ActionListener
Checkbox	チェックボックス	ItemListener
PopupMenu	選択メニュー	ItemListener
List	スクロールリスト	ActionListenerかItemListener
Choice	メニュー項目	ItemListener
Scrollbar	スクロールバー	AdjustmentListener
TextArea	テキスト領域	TextListener
TextField	テキストフィールド	TextListener

このような構造物は、グラフィックを用いて、ユーザとのインターフェース（Graphic Uses Interface：GUI）を行なう部品（Component）になっています。アプレットを定義する際に利用するインターフェースとしては、ActionListner以外に、複数の項目から選択させるためのItemListenerや、文字入力に逐一対処できるためのTextListnerなどが用いられます。

このようなGUI部品をユーザが操作することによって発生する通知は、高レベルのイベント（Highlevel Event）と呼ばれています。ActionListenerなどのインターフェースを利用すれば、通知されたイベントに対して自動的に、登録してあったオブジェクト（インスタンス）のメソッドが呼び出される仕組みになっています。それぞれのインターフェースに対しては、次のようなメソッドが呼び出されることになっています。

インタフェース	呼び出されるメソッド
ActionListener	actionPerformed(ActionEvent *ae*)
ItemListener	itemStateChanged(ItemEvent *ie*)
AdjustmentListener	adjustmentValueChanged(AdjustmentEvent *ae*)
TextListener	textValueChanged(TextEvent *te*)

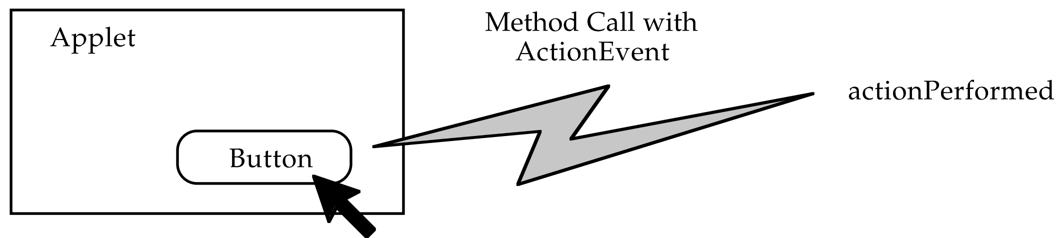

図13-6　ボタンが押されることによってactionPerformedが呼び出される

１３－４－２．ボタンを用いる

以降、上に挙げたGUI部品については、後の章で詳しく紹介いたします。ここでは、まずSwingのボタンについて、アプレットでどのように扱っていくか紹介します。

★画面上へのボタン（オブジェクトの１つ）の作り方

ボタンのためのクラスは、Buttonクラスです。Buttonクラスのオブジェクトを保持する変数を用意して、アプレットの描画領域上に配置するのですが、更に、ボタンが押されたときに、どのオブジェクトが反応するのかを予め登録しておかなければなりません。以下のようなステップになります。

1. ボタンの状態を保持する変数を用意する（多くはインスタンス変数として）

```
Button          変数名;
```

※たいていは、インスタンス変数として用意しておきます。

2. その変数に名前つけて初期化する

```
変数名 = new Button( "ボタンの名前" );
```

※ここでのボタンの名前の部分は、表示されたときに使われます。

3. そのボタンが押されたときに反応するオブジェクトを登録する

```
変数名.addActionListener( this );
```

※反応するオブジェクトは、アプレット自身なので**this**を指定しています。

4. ボタンをアプレット上（画面の描画領域）に配置する。

```
add( 変数名 );
```

※きちんと指定された場所に表示することはできません。だいたいは、上部の中央にボタンが表示されます。

例えば、以下のプログラムの断片は、OKと表示されたボタンをアプレット上に配置します。アプレット自身が、ボタンが押された場合に対処するという指定をしています。

```
Button  mybutton;
mybutton = new Button( "OK" );
mybutton.addActionListener( this );
add( mybutton );
```

この例では連続的に書いていますが、ボタンの状態を後で変化させたいような場合は、ボタンを参照する変数

はインスタンス変数として宣言します。後で参照しない場合は、メソッドなどのローカル変数として定義しても構いません。その場合でも、ボタン自体は表示されたままで残っています。

これらの一連のステップは、通常はアプレットがWebに読み込まれたときに実行するのが一般的です。何度もボタンを登録したり、配置したりする必要はないからです。そのため、ボタンオブジェクトを作って、配置するまでは、initメソッドの中で実行するのが定石です。

なお、ここでは、ボタンが１つなので、ボタンによって引き起こされるアクションの登録はしていません。複数のボタンがあるときについては、後で説明しましょう。

★四角形を移動させるアプレットの例

ボタンを表示させ、ボタンが押される度に四角形を下に移動させるアプレットです。インスタンス変数yが表示される四角形のy座標を示しています。

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

public class BoxMove extends Applet implements  ActionListener  {
        Button  button;              //  ボタンを保持するインスタンス変数
        int     y;                   //  ｙ座標を保持するインスタンス変数

        public void init() {                              // 最初に実行されるメソッド
                y = 0;                                    //  ｙ座標を0にする
                button = new Button( "Go!" );             // Go!という名前でボタンを作る
                button.addActionListener(  this  );       // アプレット自身がイベントを受け取る
                add( button );                            // 描画領域上にボタンを配置
        }

        public void paint( Graphics g ) {                 // 描画を行なうメソッド
                g.drawRect( 10, y, 20, 20 );   //  ｙ座標の位置に四角形を表示する
        }

        public void actionPerformed( ActionEvent e  ) {        // ユーザの行為に反応するメソッド
                y += 10;                                       // y 座標を更新する
                repaint( );                                    // Webブラウザに再描画を要求する
        }
}
```

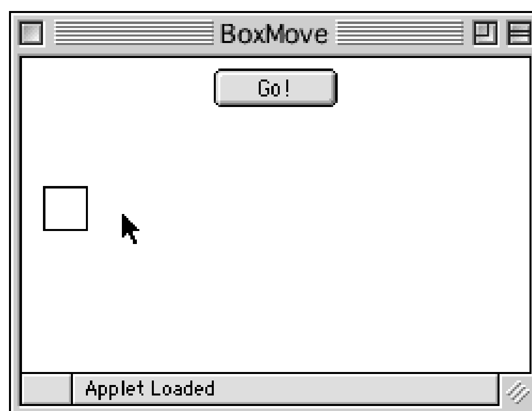

図13-7　BoxMoveアプレットの実行例

★インスタンス変数に注目

値の変化に注目して下さい。クラスの名前を定義した直下で宣言されているインスタンス変数は、メソッドとは関係なく、アプレットと共に存在し続けます。

```
Button    button;          // ずっとボタンの情報を持っている
int       y;               // ずっとy座標を持っている
```

これらのインスタンス変数がアプレットの状態を示しているとも言えます。そのような意味で、状態変数とも呼ばれることがあります。

１３－４－３．MVCプログラミング

ユーザの応答をアプレットのプログラミングをよく理解するために、MVCと呼ばれるプログラミング手法を知っておきましょう。

★モデルとビューとの切り分け

1973年Xerox Palo Alto研究所でALTOワークステーションの最初のマシンが開発されました。このマシンは、現在普及しているパーソナルコンピュータの原型をなすものです。ウィンドウシステムを持ち、マウスで情報を制御することができ、ネットワークで他のコンピュータと接続されていました。Smalltalkは、ALTOワークステーションで稼働するプログラミング言語環境でしたが、ここで後のウィンドウシステムのプログラミング方法に影響を与えるMVC（Model-View-Controller）という概念が確立しました。ALTOワークステーションは、Alan KayのDynabook（東芝のその名前を借用したコンピュータとは関係がありません）の構想の実現のために作られたもので、６歳～１３歳ぐらいの子供にコンピュータを使わせて、コンピュータのユーザインターフェースやプログラミングの評価をしました。もちろん、Smalltalkも子供たちが利用してさまざまなアプリケーションを作っていました。そのいくつかは、現在使われている主要なソフトウェアの原型となるものでした。

Alto Workstation &
Smalltalk & Children

図13-8　Altoワークステーションと子供たち

MVCのプログラム方法では、表現したい対象をモデルと呼んでいます。また、それが実際に画面に描画された表象を指してビューと呼びます。そしてユーザと応対し、ビューやモデルを制御する機構をコントローラと呼んでいます。このようにしておけば、モデルは１つでもあって、複数のビューを持つことができますし、各ビューに対してコントローラを用意することもできます。

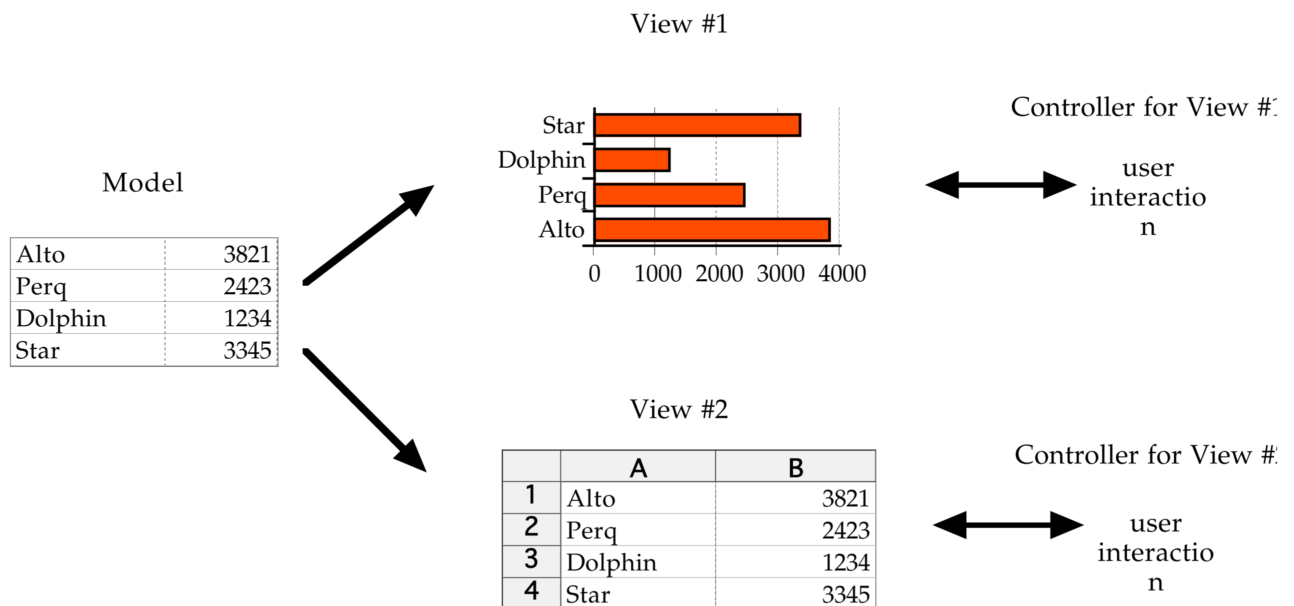

図13-9　MVCの例

この図の例では、モデルはある関連づけられたデータの集合を表しています。ビューでは、それをグラフにして表すこともできますし、表計算ソフトウェアのように表すこともできます。それぞれのビューには、対応するユーザインターフェースを実現するためのコントローラが用意されています。

ビュー上で、ユーザが何かの情報の更新を行なえば、コントローラからモデルに対して変更が行なわれます。モデルの更新は、複数のビューが存在すれば別のビューにも反映されることになるでしょう。例えば、上の例では表計算のセル（１つのデータを表す枠のことを指す）上で、データが変更されれば、グラフにもその変更が反映されることになります。

★アプレットにおけるMVCプログラミング

Javaにおいて、MVCはどのように実現するのでしょうか？アプレットにおいては、ユーザとの対応をするコントローラは、ActionListenerインターフェースを付加することにより実現しています。なお、現在はApplet自身がコントローラも兼ねていますが、別クラスに独立させることも可能になっています。

では、モデルとビューはどのように切り分けるのでしょうか？四角形を移動させるプログラムの例を取って考えてみましょう。このプログラムにおいては、特に表現したいデータというものはありませんが、強いてモデルとビューに分けて考えると、インスタンス変数のyの値を四角形の縦方向の座標として表示したプログラムと捉えることができます。

モデル：　　　インスタンス変数yが保持する整数値
ビュー：　　　整数値に対応した場所に表示される四角形とボタン
コントローラ：　ボタンが押されたときに行なわれる四角形の移動

このように考えれば、別のビューを想定することも可能になってきます。たとえば、ビューを整数値に対応したカラーの表示（塗りつぶされた四角形で表示される）としましょう。paintメソッドは次のように書き換えることができるでしょう。

```
public void paint( Graphics g ) {
        g.setColor( new  Color(  y, 0, 0 )  );          // yの値は赤色の強さを表す
        g.fillRect( 10, 10, 100, 100 );
}
```

コントローラは変わっていませんが、ビューが上のように入れ換えられた場合、ボタンが押されたときに色が

変化することになります。このように、ユーザとの応対を行なう複雑なアプレットを設計するときは、モデルとビュー、およびコントローラに分けてプログラミングを行なうことが重要になってきます。

多くのアプレットにおいては、次のようにMVCを実現しています。

・モデルとして保持される情報は、アプレットのインスタンス変数として宣言します。
・ビューはpaintメソッドで実現します。何故なら、描画領域にグラフィックスを描けるのはpaintメソッドだけだからです。
・コントローラは、ユーザの応答を担当するメソッド（例えばactionPerformed）で実現します。コントローラは、ユーザの応答に従って、モデルであるインスタンス変数が持つ値を書き換えてから、再描画するように要求します。

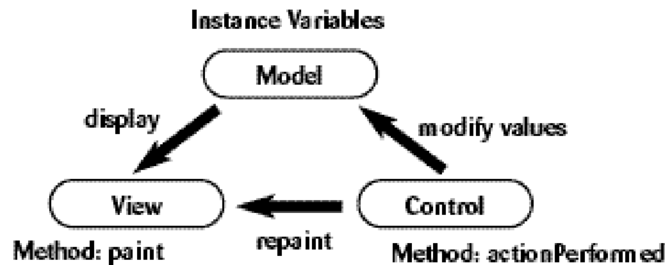

図13-10　アプレットにおけるMVC

１３－５．複数のボタンの識別方法

Javaでは、アプレット上に表示されたボタンから、コントローラ（すなわちリスナーとして登録されたオブジェクトのメソッド）に対してアクションイベント（ActionEvent）が送られる仕組みになっています。ボタンが一つであれば、単純にボタンに対応した操作をすれば良かったのですが、ボタンが複数ある場合は、まずは、どのボタンが押されたかを識別しなければなりません。これには、２つの方法があります。

１３－５－１．イベントの発生源を識別する方法

actionPerformedメソッドが受け取ったActionEventのオブジェクトには、イベントの発生源を調べるためのgetSourceメソッドが組み込まれています。他のクラスのイベントも、すべてはEventObjectというクラスをスーパークラスとして持っていますので、このメソッドを利用することができます。

```
イベント.getSource( )
```

このようにすると、以下のようにイベントを発生したオブジェクトを返してくれます。

```
Object   source = ae.getSource( );
```

このオブジェクトをインスタンス変数で保持していたボタンなどのオブジェクトと比較することによって、どのボタンがそのイベントを発生させたのか調べることができます。以下のような感じで記述していきます。

```
public void actionPerformed( ActionEvent  e ) {
        if  ( e.getSource( ) == あるボタン ) { そのボタンによってなされること }
        else  if ( e.getSource( ) == 別のボタン  ) { 別のボタンによってなされること }
        repaint( );
}
```

★四角形を上下に動かすアプレットの例

既出の四角形を動かすBoxMoveプログラムを上下方向に動かすボタンを持つように改修してみました。upとdownという変数に、それぞれ上下用のボタンを持たせています。

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

public class BoxMove extends Applet  implements  ActionListener {
        Button   up, down;
        int      y = 0;

        public void init( ) {
                up = new Button( "Up" );
                up.addActionListener( this );
                add( up );

                down = new Button( "Down" );
                down.addActionListener( this );
                add( down );
        }

        public void paint( Graphics g ) {
                g.setColor( Color.gray );
                g.fill3DRect( 10, y, 20, 20, false );
        }

        public void actionPerformed( ActionEvent  e ) {
                if  ( e.getSource( ) == up )  {  y -= 10; }
                else  if ( e.getSource( ) == down  ) { y += 10; }
                repaint();
        }
}
```

１３−５−２．他のコンポーネントとボタン

他のコンポーネントとして、LabelとChoiceを使ってみましょう。Labelは、受け身的なオブジェクトで、コンポーネントの中に文字列を表示するだけの機能しかありません。Labelは、drawStringと変わりがないように思えるのですが、どのような場合に使うのでしょうか。Labelの特徴として、次のようなこと利点があります。

・文字列の大きさに応じてコンポーネントの大きさを調整してくれる
・レイアウトマネージャが自動的にレイアウトしてくれる
・コンポーネントに共通な、色を変えたり、フォントを変えたりするメソッドが用意されている

そのため、情報だけを表示する場合は、drawStringではなくて、Labelを使うことが多いのです。簡単にLabelを使う方法は、次の通りです。

```
Label   変数 =  new  Label(  "ラベルとして表示する文字列" );
add(  変数  );
```

Labelに表示する文字列は、後からsetTextメソッドを使って変えることができます。しかし、Label自体は最初にコンポーネントが作られた段階で大きさが計算されています。後から表示する文字列を変えることを計画している場合は、上記のコンストラクタで作成する際の文字列は、予め大きめにしておきます。以下の例は、ボタンを作る前に、ボタンを押すことを促すラベルを用意するための記述になっています。

```
Label   mylabel  = new  Label( "右のボタンを押してください" );
add(  mylabel  );
```

Choiceは、ポップアップ型のメニューで、複数の項目から１つの項目を選択させるために用います。次のような形で作っていくことができます。

```
Choice   変数 =  new  Choice(  );
変数.add(  "メニューの選択項目"  );             //  項目の個数分だけ、何回も行なう
add(  変数  );
```

Choiceの項目の選択の反映については、２つの方法が考えられます。この２つの方法でサンプルのプログラムを作成していきましょう。

１．ボタンと連動して、ボタンが押された時点で、Choiceで選ばれたメニューの項目が反映される
２．Choiceでメニューの項目を選ぶたびに、その選択が反映される

１の場合は、これまでのButtonとActionListnerを使って、actionPerformedメソッドの中でどの項目が選択されたか調べることで実現できます。何番目の項目が選択されたかを調べるには、getSelectedIndex()というメソッドを用いることができます。これは、項目が追加された順番に、0番から始まり、何番目の項目が選択されたかを整数として返してくれるメソッドになっています。なお、selectメソッドは番号で項目を選択しています。

以下のアプレットのプログラムは、四角形の色をメニューに応じて変化させるものです。ボタンを用意してボタンが押された時点で色を反映させるものです。

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

public class ColorSelector extends Applet implements ActionListener {
        Choice  colorselect;            int         selected = 0;
        Label     mylabel;
        Button   button;

        public void init( ) {
                mylabel = new Label( "Select Color and Push Button" );
                add( mylabel );
                colorselect = new   Choice( );
                colorselect.add(  "Red" );
                colorselect.add(  "Green" );
                colorselect.add(  "Blue" );
                colorselect.select( 0 );
                add( colorselect );
                button = new Button( "Change" );
                button.addActionListener( this );
                add( button );
        }
        public void paint( Graphics g ) {
                g.setColor( ( selected == 0 ) ? Color.red : ( selected == 1 ) ? Color.green : Color.blue );
                g.fillRect( 20, 40, 100, 100 );
        }
        public void actionPerformed( ActionEvent e ) {
                selected = colorselect.getSelectedIndex( );
                repaint( );
        }
}
```

２の場合は、Choiceの選択が変わるたびに、それを反映させるメソッドを起動させなければなりません。この場合には、ItemListenerというインターフェースと、itemStateChangedというメソッドを用意し、initメソッドの中で、対象となるChoiceのオブジェクトに対して、addItemListenerで選択変更時に起動されるリスナーオブジェクト（そのイベントを受け取るオブジェクト）を指定する必要があります。

以下のアプレットのプログラムは、先程のプログラムをボタンなしで２の場合として書き直したものです。

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

public class ColorSelector extends Applet implements ItemListener {
        Choice colorselect;        int    selected = 0;
        Label    mylabel;

        public void init( ) {
                mylabel = new Label( "Select Color" );
                add( mylabel );
                colorselect = new  Choice( );
                colorselect.add( "Red" );
                colorselect.add( "Green" );
                colorselect.add( "Blue" );
                colorselect.select( 0 );
                colorselect.addItemListener( this );
                add( colorselect );
        }
        public void paint( Graphics g ) {
                g.setColor( ( selected == 0 ) ? Color.red : ( selected == 1 ) ? Color.green : Color.blue );
                g.fillRect( 20, 40, 100, 100 );
        }
        public void itemStateChanged( ItemEvent e ) {
                selected = colorselect.getSelectedIndex( );
                repaint( );
        }
}
```

今回は、いくつかの項目のレイアウトを標準のレイアウトを使って配置しました。これは、FlowLayoutと呼ばれ、自動的に位置を計算してくれるものです。アプレット上部から中央揃えでコンポーネントを配置してくれます。それ以外のレイアウトや、自分でコンポーネントの大きさや、位置を指定する方法は、コンポーネントの章を参照してください。

１３－６．課題

13-1.
ボタンを使った例について、自分でアプレットを作成し、実際にボタンを押して、下方向に動くかどうか確かめてみなさい。その後、ボタンを２つ使った例に改修して、同じ事を行いなさい。

13-2.
以前で出てきたカラーテーブルアプレットを改良して、緑色成分に関して、強さを+16するボタンと強さを-16するボタンを付け足して、ボタンが押される度にカラーテーブルを書き直すように改良しなさい。また、ボタンを押しすぎて緑の成分の強さが256以上になったりする場合や、0未満になったりする場合にも対処しなさい。

ヒント：緑の成分の強さを矯正する部分もactionPerformedメソッドの中に書き加えます。

13-3.
四角形が上下に動くアプレットの例を、更にボタンを２つ付け加えて左右にも動くように改良してみなさい。

また、描画領域の端からはみ出たら、もう一方の端に表示されるようにしなさい。

ヒント：x座標を持つインスタンス変数を用意します。

13-4.
以前出てきたリサージュ図形を描くアプレットを改良して、２つのボタンを用意し、１つのボタンが押されたら、cosの方の倍率を、もう１つのボタンが押されたら、sinの方の倍率を上げるように改良しなさい。なお、倍率が10倍を超えたら、1倍に戻すようにしなさい。

13-5.
上記の13-3の課題に付け加えて、色を変える為のChoiceメニューを追加しなさい。

13-6.
上記の13-5の課題に付け加えて、更に左右の回転ボタン、縮小ボタン、拡大ボタンを用意し、左右の回転は、それぞれ10度ずつ、縮小・拡大は、10%ずつに行なうようにアプレットを改良してみなさい。なお、回転・拡大・縮小は、四角形の重心を中心として行なうようにします。

ヒント：四角形は、４つの頂点の各座標を配列で表現し、drawPolygonで描きます。そして、グラフィックスと計算（２）の章で学んだアフィン変換を使います。