Chapter 18. オブジェクトの配列、ファイルからの入力とリスト

１８－１．オブジェクトの配列

１８－１－１．オブジェクトの配列を持つ変数

オブジェクトの配列を宣言するときは、次のように型名の替わりにクラス名を用います。上の方の書式は、オブジェクトの配列を宣言するだけです。下の方の書式は、オブジェクトの配列の実体を用意するものです。

▼オブジェクト配列の宣言の書式

```
クラス名     変数 [ ] ;
クラス名     変数 [ ]  =  new   クラス名[  配列のサイズ  ] ;
```

たとえば、次の２つの配列は、それぞれカラーとボタン用の配列となります。

```
Color   colors  [ ]  =  new   Color[ 20 ];                    //  20個のカラーオブジェクト用の配列
Button   operations [  ]  =  new   Button[  10 ];  //  10個のボタンオブジェクト用の配列
```

後は、通常の配列のように用いることができます。配列の要素を指定する場合は、通常のオブジェクトを参照している変数のように用いることができます。

```
colors[  3  ]  =  new  Color(  10, 30, 40 );                    // 3番目の要素にカラーオブジェクトを生成
g.setColor(  colors[ 3 ] );
operations[ 4 ]  =  new  Button( "Proceed" );                // 4番目の要素にボタンオブジェクトを生成
operations[ 4 ].addActionListener( this );                    // アクションリスナーを設定
add( operations[ 4 ] );
```

上の例をみてもわかるように、オブジェクトの配列は、オブジェクトそのものを生成したのではありません。**new**文では配列を１つ生成にしただけに過ぎません。ですから、配列の要素となる個々のオブジェクトは、その都度繰返しなどを使って**new**文を使ってそれぞれを生成していく必要があります。これが、通常の整数などの配列と異なる点です。

★ボタンの配列を作る

ボタンの配列を使ってみましょう。次のアプレットは、カラーの配列とボタンの配列とを用いています。ボタンが押されたら、そのボタンに対応する色を用いて四角形を塗りつぶすようなプログラムになっています。変数*carray*は、カラーの配列です。変数*barray*がボタンの配列になっています。変数*current*が、現在選択されている色の番号（インデックス）を保持しています。

initメソッドでは、配列のそれぞれの要素に、ボタンを**new**文で生成して割り当てています。配列の要素である個々のオブジェクトはこのように１つずつ生成しなければならないことに注意してください。ボタンには、例えばColor 10というようなラベルがつけられます。各ボタンには、インデックスと同じ番号がコマンドとして設定されています。ボタンが押されたときに呼ばれるactionPerformedメソッドでは、押されたボタンのコマンドを解析して、それを整数に変換しています。文字列を整数に変換するInteger.parseIntメソッドは、後の章で詳しく説明します。その整数が変数*current*に代入されて、色の番号を選択するために用いられています。

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

public class ButtonArray extends Applet implements ActionListener {
        Color     carray [ ] = { Color.red, Color.green, Color.blue, Color.magenta,
```

```
                    Color.cyan, Color.yellow, Color.pink, Color.orange,
                    Color.white, Color.black, Color.gray, Color.darkGray,
                    Color.lightGray };
    Button  barray [ ] = new Button[ carray.length ];
    int     current = 0;

    public void init( ) {
        for ( int i=0; i<barray.length; i++ ) {
            barray[ i ] = new Button( "Color "+ i );
            barray[ i ].addActionListener( this );
            add( barray[ i ] );
        }
    }

    public void paint( Graphics g ) {
        g.setColor( carray[ current ] );
        g.fillRect( 100, 100, 100, 100 );
    }

    public void actionPerformed( ActionEvent e ) {
        for ( int i=0; i < barray.length; i++ ) {
            if ( e.getSource( ) == barray[ i ] ) { current = i;  break; }
        }
        repaint( );
    }
}
```

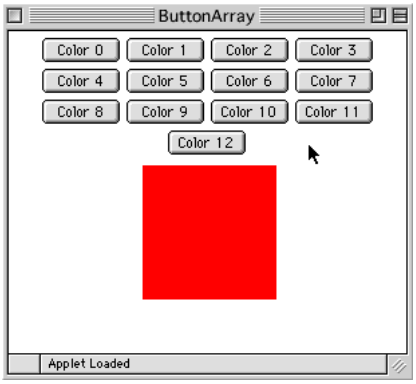

図18-1　ボタン配列アプレットの実行例

１８－１－２．オブジェクトの配列と初期化

オブジェクトを使って関連する情報をまとめあげることは本当に風に役に立つのでしょうか？配列を使った場合を考えてみると、それが顕著になります。たとえば、点の配列を考えてみましょう。オブジェクトを使わなければ、各点のx座標やy座標を別々の配列として持たなければなりませんでした。しかし、Pointクラスのオブジェクトを使えば、オブジェクトの配列を用意すれば、各点を表すことができます。x座標やy座標は、配列の要素であるオブジェクトのインスタンス変数を参照すれば求めることができます。

★オブジェクトの配列に対するインスタンス変数の参照の仕方

オブジェクトが配列として用意されているときに、各要素のオブジェクトのインスタンス変数にアクセスするときは、要素を指定してからドットで区切って指定します。

▼配列の要素のオブジェクトのインスタンス変数を参照する書式

配列名 [インデックス].インスタンス変数名

たとえば、次のような10個の要素を持つPointクラスのオブジェクトが次のようなプログラムの断片によって、用意されているとしましょう。

```
Point  points [ ]= new  Point[ 10 ];
for ( int i=0; i< points.length ;  i++ ) {
        points[ i ] = new Point(  i * 10 % 3  ,  i % 10 * 20  );
}
```

この配列の要素であるオブジェクトのインスタンス変数を参照してみましょう。

```
points[ 0 ].x = 45;
points[ 6 ].x = points[ 3 ].x + 12;
System.out.println( "Fifth point  x-axis: " + points[ 5 ].x + "y-axis:" + points[ 5 ].y );
```

さらに繰返しを使って、各点を線で結んでみましょう（アプレットのpaintメソッドの中で実行されていて、描画領域を仮パラメータ変数*g*で受け取っているとします）。

```
for ( int i=0; i< points.length - 1 ;  i++ ) {
        g.drawLine(  points[ i ].x ,  points[ i ].y, points[ i + 1].x,  points[ i + 1 ].y );
}
```

ここでちょっとしたコツを紹介しましょう。上の繰返しですと、最初（０番目）の点と最後（９番目）の点を結んでくれません。この２つの点を結んで閉じた形にするには、余りの演算を使って次のようにします。どうしてうまくいくかは、考えてみてください。

```
for ( int i=0; i< points.length;  i++ ) {
        g.drawLine(  points[ i ].x ,  points[ i ].y,
                points[ (i + 1) % points.length ].x,  points[( i + 1) % points.length ].y );
}
```

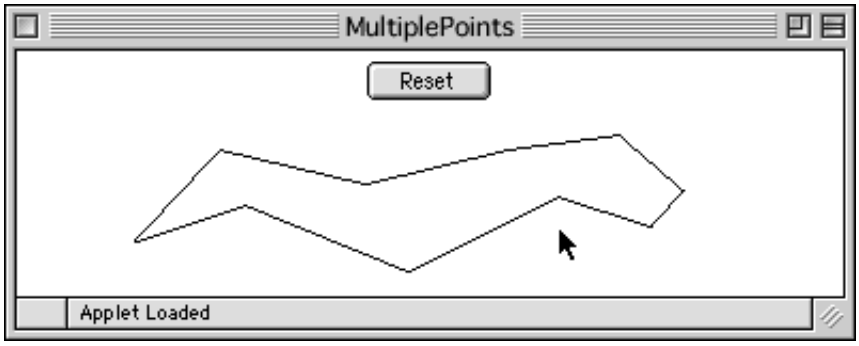

図18-2　閉包になるように点を結んだ例

★初期化の仕方

オブジェクトの配列の要素をフィールド（インスタンス変数）を含めて初期化したいときがあります。このときは、配列と同様に波括弧{ }を使うことになります。

```
Point     myPoints  [ ] = { new Point( 33, 22 ),  new Point( 22 , 11 ),  new Point( 55, 19 ) };
```

これは、以下の記述と等価です。注意しなければならないのは、配列を**new**演算子で作成した後、更に個々の要素を**new**演算子で作成しなければならないことです。配列の生成と個々のオブジェクトの生成は別であるということを再認識しましょう。

```
Point     myPoints [ ] = new Point [ 3 ];
myPoints[ 0 ] = new Point( 33, 22 );
```

```
myPoints[ 1 ] = new Point( 22 , 11 );
myPoints[ 2 ] = new Point( 55, 19 );
```

また、Java 2以降は、配列の宣言と初期値の代入を別々に行なうことができます。これは、以前に説明しました。初期値代入をしたい場合は、**new**演算子を利用することになります。

```
Point     myPoints;
myPoints = new Point  [ ] { new Point( 33, 22 ),  new Point( 22 , 11 ),  new Point( 55, 19 ) };
```

★AWTのオブジェクトを使った描画処理

Rectangleクラスのオブジェクトの配列を使って、最大50までのそれまで描いた四角形を覚えておくことができるようなアプレットを作ってみましょう。アプレットのインスタンス変数で、この配列を確保しておきます。あとは、マウスで四角形をユーザに描いてもらい、描き終わったら配列の要素に登録していきます。マウスのドラッグの部分などは、第１１章に出てきたプログラムとまったく同じです。マウスが離されたときの部分だけを追加しています。マウスが離されたときは、そのときの座標を元にRectangleオブジェクトを作り、要素として代入していきます。

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

public class MultipleMouseDrag extends Applet
        implements ActionListener, MouseListener, MouseMotionListener {
    Rectangle       rectangle [ ] = new Rectangle[ 50 ];
    int             count = 0;
    int             startx, starty, endx, endy;

    public void init( ) {
        addMouseListener( this );
        addMouseMotionListener( this );
        Button  button = new Button( "Reset" );
        button.addActionListener( this );
        add( button );
    }

    public void paint( Graphics g ) {
        for ( int  i = 0;  i < count ;  i ++ ) {
            g.drawRect( rectangle[ i ].x,  rectangle[ i ].y,
                    rectangle[ i ].width, rectangle[ i ].height );
        }
        g.drawRect( smaller(  startx, endx ), smaller(  starty, endy ),
                    difference(  startx, endx ), difference(  starty , endy ) );
    }

    public void mousePressed( MouseEvent e ) {
        startx = endx = e.getX( ); starty = endy = e.getY( );  repaint( );
    }
    public void mouseReleased( MouseEvent  e ) {
        if  (  count < 50 ) {
            rectangle[ count ++ ] = new  Rectangle(
                    smaller(  startx, endx ), smaller(  starty, endy ),
                    difference(  startx, endx ), difference(  starty , endy ) );
        }
    }
    public void mouseDragged( MouseEvent e ) {
        endx = e.getX( ); endy = e.getY( ); repaint();
    }
    public void actionPerformed( ActionEvent  e  ) {
        startx = starty = endx = endy = 0;  count = 0;  repaint( );
    }
```

```
    int smaller( int w, int u ) { return ( w < u ) ? w : u ; }  // 小さい方を返します
    int difference( int w, int u ) { return ( w > u ) ? w - u : u - w; }        // 差を返します

    public void mouseClicked( MouseEvent e ) { }
    public void mouseEntered( MouseEvent e ) { }
    public void mouseExited( MouseEvent e ) { }
    public void mouseMoved( MouseEvent e ) { }
}
```

プログラムでは、smallerとdifferenceという２つのメソッドを定義して、条件分岐をする手間を省いています。そのため、短くプログラムを記述することができています。また、ボタンを１つ用意して、押されたら、それまで覚えていた内容を全部リセットしています。

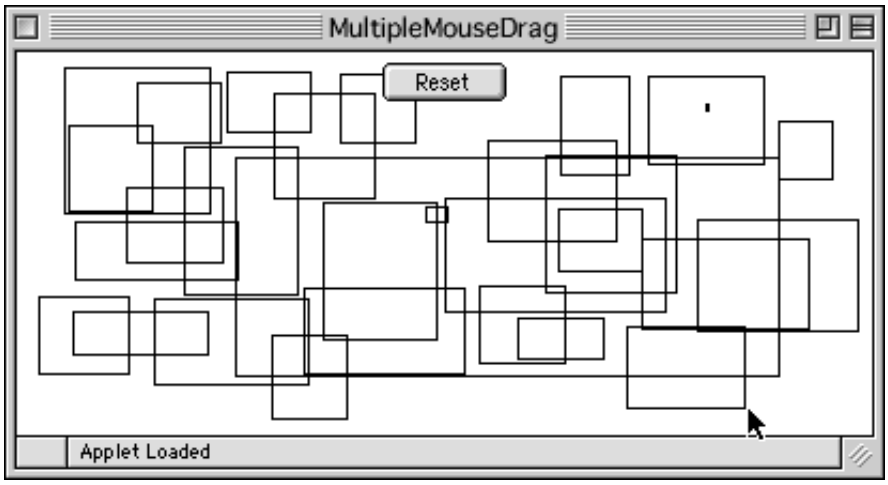

図18-3　描いた四角形を覚えておくアプレット

１８－２．文字列と配列

１８－２－１．文字列を要素とする配列

配列の要素が文字列から構成されるような配列を定義することができます。個々の要素が１つの文字列になっています。配列は、個々の文字列を複数まとめて管理することができるのです。これは便利！

```
String [ ] festival = { "たこ焼き", "焼きそば", "お好み焼き", "金魚すくい", "水飴" };
```

上のように配列に初期値代入を使って、複数の文字列がそれぞれ要素として代入されているとします。このときに、たとえば、次のようにして個々の文字列を一行ずつ表示させることができます。

```
for ( int i=0; i < festival.length ; i++ ) {
        System.out.println( festival[ i ] );
}
```

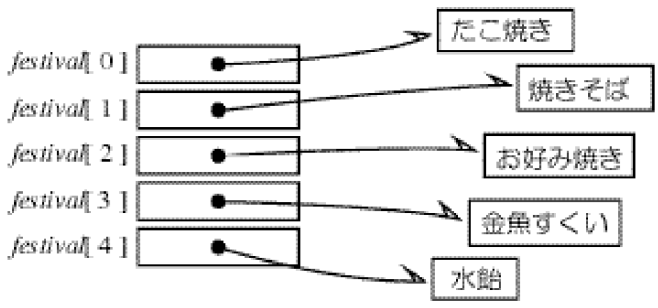

図18-4　文字列を要素とする配列

★個々の文字列を変更する

次のように文字列を要素とする配列が定義されているとしましょう。

```
String    [ ] members = { "しんご", "たくや", "ごろう", "つよし", "まさひろ" };
```

たとえば、最後の要素を別の文字列にするには次のように記述します。インデックスの4の部分は、*members*.length - 1と記述した方がどのような場合にも対処できると思われます。

```
members[ 4 ] = "ふみや";                    // 歌唱力を考慮した結果か？
```

あるいは、最初の要素と最後の要素を入れ替えてみましょう。同時に入れ替えはできませんから、一時的に入れ替えるための文字列を保存するための変数tempを用意しています。

```
String    temp = members[ 0 ];                    // temp ← "しんご"
members[ 0 ] = members[ members.length - 1 ];     // members[ 0 ] ← "まさひろ"
members[ members.length - 1 ] = temp;             // members[ 4 ] ← "しんご"
```

★アニメーションを見せる

次のように画像ファイルが名前を変えて一杯あったときに、文字列を要素とする配列を使えば、次々とロードして、アニメーションのように見せることができます。個々の要素がファイル名を示しているからです。画像ファイルは、GIF形式のファイルなのでしょう。必ず、".gif"というな名前で終わっているとします。また、それらのファイルはアプレットと同じフォルダに置かれていると仮定しています。

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

public class MeloAnimation extends Applet {
        String  melostatus  [ ] = { "sitdown", "hand", "face1", "face2", "face1", "face2",
                                    "scratch", "sitdown", "notice", "eat1", "eat2",
                                    "eat1", "eat2", "eat3" };
        public void paint( Graphics g ) {
            for ( int  i=0 ; i < melostatus.length ; i ++ ) {
                try {
                        Image   melo = getImage(  getCodeBase( ),  melostatus[ i ] + ".gif"  ) );
                        Thread.sleep( 500 );                    // 0.5秒やすみ
                        g.drawImage( melo, 0, 0, this );
                } catch(  Exception error ){   System.err.println( error ); }
            }
        }
}
```

図18-5　アニメーションに使われているGIFファイル

１８－２－２．文字列と単語リスト

★１つの文字列から単語リストに切り出す

１つの文字列を英語の１文と考えて、それを単語リストに分解することを考えましょう。単語は、最大で20個までとします。単語と単語の間は、１個の空白で区切られているとします。次のプログラムでは、まず*wordList*という文字列を要素とする配列を宣言して、20個分の文字列を参照できるようにしています。文字列*source*に入っている文の単語は、ここに切り出されていきます。単語の数を数えるための変数*wordcount*を用意しています。文字列*source*の各文字へのインデックスを変数*start*と*end*が保持しています。

```
public class StringToWord {
        public static void main( String [ ] args ) {
                String   source = "Can you tell me how to get to the airport";
                String   wordList [ ] = new String[ 20 ];

                // 単語リストに分解する
                int wordcount = 0;
                int start, end;
                for ( start=0; start<source.length( ); start = end + 1 ) {
                        for ( end = start; end < source.length( ) ; end++ ) {
                                if ( source.charAt( end ) == 32 ) { break; }
                        }
                        wordList[ wordcount ] = source.substring( start, end );
                        wordcount ++ ;
                }
                // 分解した単語リストをすべて表示する
                for ( int i = 0 ; i < wordcount ; i++ ) {
                        System.out.println( wordList[ i ] );
                }
        }
}
```

一つの単語を取り出すには、*start*の位置から始めて、*end*の位置に空白があるかどう内側の繰返しで走査しています。なければ、*end*の位置を後ろに一つずつずらしていきます。空白が見つかったら、内側の繰返しを抜け出します。そして、*start*の位置から*end*の位置の手前までの部分的な文字列をsubstringメソッドで抜き出して、それをwordListに追加しています。外側の繰返しで、次の*start*位置は*end*の次の位置にして、同じことを行なうようにします。最終的に、文字列の最後までこれを繰り返すと単語リストに分解することができます。次の図は、“tell”という単語を取り出すときの*start*と*end*の値を示しています。

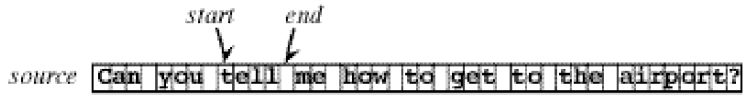

図18-6　tellを取り出すときの*start*と*end*

★StringTokenizerクラスを使って分解する
たとえば、ユーザに何かを入力して貰う場合でも１行の１つの情報を入力して貰うような場合もありますが、数値データなどを１行に１レコード分記述したいような場合があります。たとえば、drawLineなどのパラメータで、１つの線の始点と終点の座標値が次のように１行にカンマで区切って入力されているとします（。

```
78, 343, 33, 22
90, 78, 220, 221
10, 100, 150, 200
 :
```

このような１行に複数の情報が、何らかの区切り用の文字で区切られて保存されている場合には、java.util

パッケージの中に便利なStringTokenizerクラスが用意されています。これは、文字列を区切りの文字列で分解するものです。分解された文字列は、トークン（Token）と呼ばれています。次のようなコンストラクタとメソッドが利用できます。

```
new StringTokenizer( 文字列, 区切りの文字列 )     // 文字列を区切りの文字列で分解します
String    nextToken( )                          // 次のトークンを返します
int  countTokens( )                             // トークンがいくつあるか返します
boolean  hasMoreTokens( )                       // 次のトークンがあるかどうか返します
```

なお、区切りの文字列に複数の文字を記述することもできます。各文字をすべて区切りとして認識してくれます。たとえば、次のようなプログラムの断片を記述することができます。これは、スペースで区切られた文字列をトークンに分解して表示するものです。

```
import java.util.*;

StringTokenizer  tokens =  new  StringTokenizer( "This is a sample message", "  " );
while ( tokens.hasMoreTokens( ) ) {
        System.out.println( "Token is " +  tokens.nextToken( ) );
}
```

結果は次のように表示されます。

```
This
is
a
sample
message
```

１８－３．ファイルとストリーム

アプリケーションでユーザから入力されたデータを保持しておきたい、あるいはいろいろな情報を外部からプログラムに入力したいというような場合があります。特に、これまではプログラム上で定数値として表していた情報を外部から入力するようにしておけば、後でそれらの値が変わっても、いちいちプログラムを再コンパイルする必要がなくなります。このようなデータとプログラムの分離は、プログラムのデータ独立（Data Independency）として提唱されてきました。これは、１つのプログラムが、一定のデータだけではなく、別のデータにも適用できることを意味しています。

Javaの場合にはユーザからの入力を得るのと同じように、外部環境とのデータのやりとりには、一貫してストリーム（Stream）を用いることになっています。ストリームとは、指定された単位でアプリケーションにデータを入力したり、データを書き出したりするオブジェクトのことです。

１８－３－１．ReaderとWriter

JDK 1.1より、プログラムに対して入力として与えられるストリームのことをReader、プログラムの出力としてのストリームのことをWriterと呼ぶようになりました。これらのストリームは国際化対応になっていて、Unicodeベースの文字を入出力の単位として扱うことができます。Readerストリームを使えば、ファイルやWebページあるいは端末からのユーザの入力をプログラムに与えることができる。また、Writerストリームを用いれば、ファイルやユーザの端末にプログラムの処理結果を出力することができます。

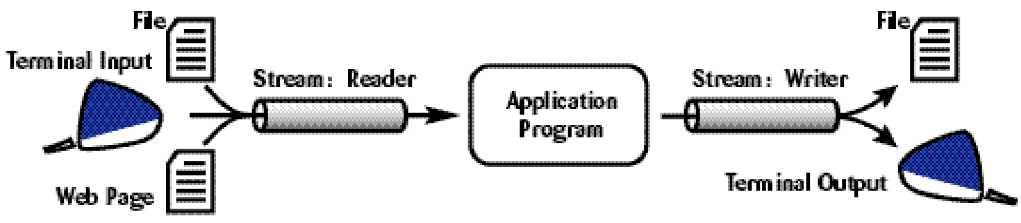

図18-7　Reader/Writerストリーム

これらのストリームに対応して、java.ioパッケージの中には、次のようなクラスが用意されています。他にも一杯クラスはあるのですが、ここでは説明に必要なクラスだけを列挙します。ここで挙げた６つのクラスは、いずれもデータをテキストファイルとして扱うものです。

Reader	入力のための抽象クラス
InputStreamReader	１文字単位で読めるクラス
BufferedReader	１行単位で読めるクラス
FileReader	ファイルから読めるクラス
Writer	出力のための抽象クラス
BufferedWriter	１行単位で出力するクラス
FileWriter	ファイルに出力するクラス
PrintWriter	文字列を出力するクラス

出力用のクラスは、BufferedWriterは、１行分を貯めてから出力するクラスです。１文字単位で出力するよりも効率的になります。２進数を直接扱うバイナリデータなどをファイルに出力するときは、FileWriterクラスを用います。テキストデータを出力するときは、printlnやprintなどのお馴染のメソッドを持ち、かつ日本語などにも対応しているPrintWriterに一旦変換してからファイルに出力ことができます。

入力用のクラスのBufferedReaderも、１行分をまとめて文字列として読み込むためのクラスになっています。InputStreamReaderやFileReaderからクラスを変換する形で、利用します。

ファイルを使うときの手順では、次のように使う前にオープンし、使った後はクローズするのが一般的です。これは、コンピュータに現在どのファイルを使っているかを管理してもらうために行なうものです。オープンは、大抵の場合は、ファイルを表現するストリームクラスのオブジェクトを作成することによって実現します。

▼ファイルを扱うときの手順

１．ファイルをオープンする
２．ファイルに対して、読む（入力）あるいは書く（出力）
３．ファイルをクローズする

それでは、まずはファイルを使ったデータの出力からみていきましょう。

１８－３－２．アプレットとストリーム

アプレットはその性質上、入出力の対象となるファイルは、クライアント上のファイルではなく、Webサーバー上に置かれているファイルになります。しかしながら、アプレットではセキュリティ的な制約から、任意のWebサーバーに対してのファイルへの入出力は許可されていません。Webサーバー上の内部のファイルを読み書きできると、第三者がアプレットを使ってWebサーバーに侵入することができるからです。

アプレットを実行するための環境としてアプレットビューワを選んだ場合には、設定としてこの制限を解除することも可能です。これは、アプレットビューワがあくまでもアプレットのテスト環境であり、ローカルなアプレットの実行を前提としているからです。一般的なWebブラウザ、たとえばInternet ExplorerやNetscape Navigatorなどでは、上記のセキュリティ的な理由から、ファイルを読み書きすることは許可されていません。

アプレットで可能なのは、元々そのアプレットが置かれていたWebサーバ上のWebページを読み出すことだけです。これは、画像ファイルや音声ファイルのロードと同様に、サーバ上のファイルを相対的に指定して行なうことができます。

★アプレットでWebページから入力を貰う

アプレットでは、URLを指定して、そこからWebページのデータをプログラムへ入力させることができます。アプレットで、ファイルに出力することはできませんが、Webサーバーにおいてある情報ファイルをデータ入力のために読み込むことは頻繁にあるでしょう。

Webページからのデータ入力の場合は、次のように行なっていきます。

１．URLを指定する。
２．指定したURLに対してストリームをオープンする
３．データを入力する
４．ストリームをクローズする

URLの指定の仕方ですが、次の３つの例は、以前画像ファイルのときに説明しました。そこで説明したように、セキュリティ上の問題から、HTMLファイルが置かれているWebサーバだけのアクセスに限りたいと思います。そのため、一般的なURLの指定はせずに、Appletのクラスファイルからの相対的な指定の方法だけを使うようにしましょう。

```
URL urladdress = new URL( getCodeBase( ), "project/infodata" );     // クラスファイル相対
```

ストリームのオープンは、URLクラスのオブジェクトは、openStreamメソッドを使うことによって行なうことができます。

```
InputStream      istream = urladdress.openStream( );
```

上記の記述のように、openStreamメソッドは、Systeminと同じInputStreamのオブジェクト（ストリーム）を返してくれます。これをBufferedReaderにするには、一旦InputStreamReaderのオブジェクトを介して変換します。これを、openStreamメソッドと一緒に行ないますと、次のような記述となります。

```
BufferedReader    br = new   BufferedReader(
        new InputStreamReader( urladdress.openStream( ) ) );
```

後は、BufferedReaderを使った入力ストリームの扱いと同じです。readLineメソッドを使って、データ入力し、入力し終わったら、closeメソッドでストリームを閉じます。

```
String line  = br.readLine( );              // 一行読み込む
br.close( );                                // ストリームを閉じる
```

★情報ファイルを配列に読み込む

さまざまなデータから構成される情報ファイルを扱います。まず情報ファイルに書かれている内容を配列に読み込んでみましょう。ここでもファイルはテキストファイルで保存されているとします。ファイル（ファイル名をinfoとします）の内容が次のような名前とその人の給与、という２行で１組になっているデータの連続で構成されているとしましょう。

▼情報ファイルの構成の例

```
Robin Milner                // １行目には人の名前が書いてある
35000                       // ２行目にはその人の給料が書いてある
Brian Smith                 // 同じように次の人の名前が書いてある
42000                       //   次の人の給与
 :                          // これが延々と続いていく
```

２行単位でファイルが構成されていると仮定しまして、これを配列に読み込んでみます。簡単にするめに、最大20個のデータを読み込むことにします。ただ単に配列に読み込んで、それを画面に表示するプログラムの断片は、次のようになります。名前用の文字列を要素とする配列と給与用の整数配列をそれぞれ用意します。カウンタを設けて、20個以上は読み込まないようにします。給与の方は、文字列として１行を読み込んだ後、

Interger.parseIntメソッドを呼び出して整数に変換しています。

```
String    name [ ] = new String[ 20 ];          // 20人分の名前を保持する配列
int       salary [ ] = new  int[ 20 ];          // 20人分の給与を保持する配列
int       count = 0;                            // 読み込んだ人数
BufferedReader br;                              //  読むためのストリーム

// 実際に読み込む部分だけ
        while ( br.ready( ) && count < 20 ) {
                name[ count ] = br.readLine();
                salary[ count ] = Integer.parseInt( br.readLine( ) );
                System.out.println( "Person: " + name[ count ] );
                System.out.println( "Salary: " + salary[ count ] );
                count++;
        }
        br.close( );
```

★テキストフィールドを２つ用意して、そこに情報ファイルから読み込んだ内容を表示させる

２行を１人の情報として構成されているinfoという名前の情報ファイルが、Webサーバー上のHTMLファイルと同じフォルダ（ディレクトリ）にあるとしましょう。このファイルを読み込んで、最初の人物の情報（名前とその給与）を表示するようなアプレットを以下に示します。２つのテキストフィールドを用意して、１人分の名前と給与を表示しています。

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

public class InfoViewer extends Applet {
        BufferedReader br ;
        public void init( ) {
                try {
                        URL  urladdress =  new  URL(  getCodeBase( ), "info" );
                        br = new BufferedReader(
                                new InputStreamReader(  urladdress.openStream( )  ) );

                        String    name =  br.readLine( );
                        String    salary = br.readLine( );
                        br.close( );

                        TextField  namefield =  new  TextField( 30 );
                        TextField  salaryfield =  new  TextField( 10 );


                        namefield.setText( name );  add(  namefield );
                        valuefield.setText(  salary );  add(  salaryfield  );
                } catch( Exception  e ) {
                        System.err.println( e );
                }
        }
}
```

このアプレットは、最初の１人分しか表示してくれません。また、給与に関しては整数値に変換していません。テキストフィールドに表示するのに、そのまま入力した文字列を表示しています。

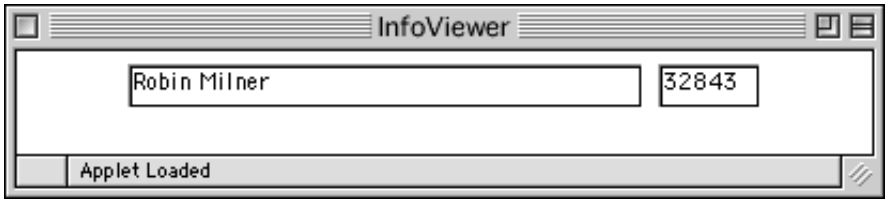

図18-8　アプレットによる情報ファイルの表示例

★ファイルの内容をテキストエリアに読みだす

次のアプレットプログラムは、テキストエリアに、infoというファイル名のファイルから読み込んだ内容を文字列として、表示するものです。１行読み出しては、appendメソッドを使って、テキストエリアに内容を追加していきます。追加するときに、改行も同時に挿入していることに注意してください。

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

public class FileViewer  extends Applet  {
        BufferedReader   br;
        TextArea    area;
        String   feed;

        public void init() {
                feed = System.getProperty( "line.separator" );
                area = new TextArea( 20, 40 );
                area.setFont( new Font( "SansSerif", Font.PLAIN, 18 ) );
                add( area );
                try {
                        URL   address = new URL( getCodeBase( ), "info" );
                        br = new BufferedReader( new InputStreamReader(
                                        address.openStream( ) ) );
                        while ( br.ready( ) ) {
                                String  line = br.readLine( );
                                area.append( line + feed );
                        }
                        br.close( );
                } catch( Exception  exc ) {
                        exc.printStackTrace( );
                }
        }
}
```

１８－４．可変長のサイズを持てるオブジェクトの集合用のクラス

配列を使った場合は、配列のサイズは固定でしたので、最初に決まった以上のサイズよりも多くの要素を持つことができませんでした。ユーザやファイルからの入力などを受ける場合など、最初に最大限のサイズがまったくわからない場合も多くあります。そのような場合のために、Listインタフェースがあります。これは、１つ１つの要素を綱で結んだような格好になっており、いくつ要素が増えても大丈夫なようになっています。このような、サイズが未決定の集合（オブジェクトの集合）を扱うための便利なクラスがjava.utilパッケージの中に用意されています。ここでは、集合を表すクラスで共通に用いられているCollectionインタフェースの機能を説明し、その中から、頻繁に使われるArrayListクラスとHashTableクラスを紹介します。

１８－４－１．Collectionインタフェース

Collectionインタフェースには、オブジェクトの集合を扱うために次のようなメソッドが用意されています。

★共通メソッド

boolean	add(Object *element*)	オブジェクトを集合に追加します
boolean	remove(Object *element*)	集合からオブジェクトを削除します
boolean	contains(Object *element*)	集合の中に、そのオブジェクトがあるかどうか
int	size( )	集合の中のオブジェクトの個数を返します
boolean	isEmpty( )	集合が空かどうか返します
Iterator	iterator( )	オブジェクトへの順序集合を返します
Object [ ]	toArray( )	集合をオブジェクトの配列に変換します
List	Arrays.asList(Object [ ])	リストにします

★イテレータのメソッド

Object	next( )
boolean	hasNext( )

１８－４－２．ArrayList

ArrayListは、配列の特徴と、要素をどんどん追加できる両方の特徴を持っています。このため、一番よく使われるリスト構造のクラスになっています。

★AbstractListインタフェースからのメソッド

void	add(**int** *index*, Object *element*)
Object	get(**int** *index*)
Object	set(**int** *index*, Object *element*)
Object	remove(**int** *index*)
boolean	equals(Object *target*)
List	subList(**int** *fromindex*, **int** *toindex*)

★独自のメソッド

int	indexOf(Object *target*)
int	lastIndexOf(Object *target*)

★ArrayListの使い方

次のアプレットは、ファイルの内容をすべて読み込み、それをボタンがおされるたびに、１行ずつテキストフィールドに表示するアプレットになっています。

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import java.util.*;

public class ListReader extends Applet implements ActionListener {
        BufferedReader  br;
        ArrayList  linelist;
        String  feed;
        Button  proceed;
        TextField  result;
        int  current = 0;

        public void init() {
                feed = System.getProperty( "line.separator" );
                linelist = new ArrayList();
                try {
                        URL  address = new URL( getCodeBase( ), "info" );
                        br = new BufferedReader( new InputStreamReader(
                                        address.openStream( ) ) );
```

```
                    while ( br.ready( ) ) {
                        String  line = br.readLine( );
                        linelist.add( line );
                    }
                    br.close( );
                } catch( Exception  exc ) {
                    exc.printStackTrace( );
                }
                proceed = new Button( "Go Next" );
                proceed.addActionListener( this );
                add( proceed );
                result =  new TextField(  60  );
                result.setText(  (String) (linelist.get( current ) ) );
                add( result );
        }

        public void actionPerformed( ActionEvent  ae ) {
                current = (current + 1) % linelist.size( );
                result.setText(  (String) (linelist.get( current ) ) );
                repaint( );
        }
}
```

１８－４－３．Hashtable

★Mapインタフェースからのメソッド

```
Object   put( Object key, Object value )
Object   get( Object key )
Object   remove( Object key )
int      size( )
Collection       values( )
```

★独自のメソッド

```
Enumeration      keys( )
Enumeration      elements( )
```

１８－５．課題

18-1.
文字列とボタンを配列として用意して、ボタンの名前を、文字列の配列を使って初期化しなさい。

18-2.
文字列を要素とする配列を使った単語リストがあるとします。この単語リストで辞書順で一番最初に出てくる単語、および辞書順で一番最後に出てくる単語はどれかを探すようなアプリケーションプログラムを作りなさい。クラス名は、FirstAndLastWordとします。

18-3.
次のように数字が１つの空白で区切られている文字列があるとします。この文字列の各数字列を整数と解釈して、整数配列に格納しなさい。整数は、最大20個とします。

```
String    sequence = "453  23  4543  972  129  5  221  876  62";
```

ヒント：１つの文字列から単語リストに切り出す例とInteger.parseIntメソッドを用います。

18-4.
TextAreaを用意し、そこで何行かユーザに入力して貰うようなアプレットを作ります。入力した後に、各行を分解して、文字列のリストに格納するようなアプレットを作りなさい。クラス名は、LineEditorとします。