Chapter １１．オブジェクトについて

１１－１．オブジェクトの生成

第３章で説明しましたが、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）と呼ぶことは前にも紹介しました。ここで、実際にJavaではオブジェクトをどのように記述することができるのか説明していきましょう。

１１－１－１．オブジェクトの生成の書式

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。Frame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  Frame( "Sample" );  // Sampleという名前で、ウィンドウ枠を作る
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。

１１－１－２．オブジェクトへのリファレンス

生成したインスタンスを、その後も参照したい場合は変数を使います。変数は、生成されたインスタンスを参照することができます。また、変数は、インスタンスが属するクラスを型とする変数として宣言されている必要があります。クラスのオブジェクトを参照するような変数を宣言するのは、次のような構文を用います。

▼書式

```
クラス名　　変数名 ;
```

たとえば、次のようにオブジェクトを参照するための変数を宣言することができます。

```
Color      frontcolor;      // Colorクラスのオブジェクトを参照する変数
Graphics   g;               // Graphicsクラスのオブジェクトを参照する変数
Frame      window;          // Frameクラスのオブジェクトを参照する変数
```

変数は、基本的な値（整数や実数など）を持つためにも使うことは従来のプログラミング言語と共通なのですが、Java言語ではこのように変数をオブジェクトを参照するためにも用いることができるのです。変数は、生

成されたオブジェクトを指し示すために用いられます。

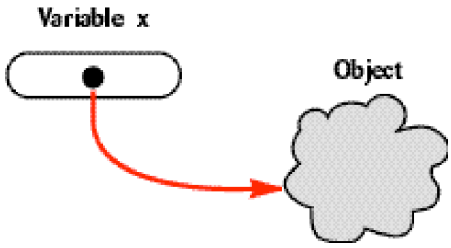

図11-1　オブジェクトを指す変数

変数に何らかのオブジェクトを参照させるためには、そのことを代入文を用いて記述する必要があります。これは、通常の代入文と同じですが、特にコンストラクタを用いて生成された新しいオブジェクトをすぐに参照したい場合は、次のように記述します。

▼変数への代入の書式

```
変数名= new　クラス名 ( パラメータ ) ;
```

たとえば、次のように記述します。それぞれ、新しく生成されたオブジェクト（インスタンス）を参照するようになります。なおRectangleは、Java2からAWTパッケージにはいっているクラスです。

```
window = new  Frame( "Sample" );          // 新しく生成したインスタンスを参照する
rectangle = new  Rectangle( 100, 50 );    // 四角形オブジェクトを参照する
```

オブジェクトの生成と変数の宣言を同じに行ないたいときは、通常の値を保持する変数と同様に初期値代入の書式を用いることができます。

▼初期値代入の書式

```
クラス名　　変数名= new　クラス名 ( パラメータ ) ;
```

以下の例は、生成されたオブジェクトを変数で参照すべく、初期値代入を行なった記述です。

```
Frame   window = new  Frame( "Sample", 400, 200 );
Graphics  g = new  Graphics( );
```

１１－１－３．オブジェクト変数の利用方法

では、生成されたオブジェクトを変数で参照することによって、何をするのでしょうか？それは、変数を通じて、そのオブジェクトが持っているフィールド（属性値）やメソッドを利用することができるのです。特に、ここではメソッド呼出しの場合について紹介しましょう。

▼変数で参照しているオブジェクトのメソッドを呼び出す書式

```
変数名 . メソッド名 ( パラメータ ) ;
```

次の例は、オブジェクトを参照する変数を通じて、Frameクラスあるいは Graphicsクラスのオブジェクトが持つメソッドを呼び出しています。

```
window.show( );                    // Frameクラスのshowメソッドを利用
```

```
gc.drawString( "Hello!", 110, 45 );                // Graphicsクラスのメソッドを利用
```

あるいは、呼び出したメソッドが何かの情報を返してくれることがあります。そのときには、次のように代入文を使って記述します。

▼情報を返してくれるメソッドを呼び出す場合

```
情報を保持する変数 = 変数名 . メソッド名 ( パラメータ ) ;
```

たとえば、次の記述は、Colorクラスのオブジェクトを指している変数*c*に対して、そのカラーの赤色成分の強さを整数として受け取っている例です。変数の宣言と代入の２行に分けて記述してみました。

```
int red ;
red = c.getRed( );
```

メソッドが返してきてくれる情報が別のオブジェクトである場合もあります。この場合には、そのオブジェクトを参照するための変数を宣言しておかなければなりません。そして、その変数に代入してやります。

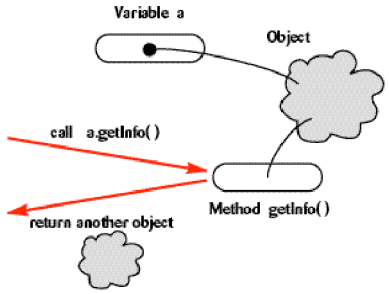

図11-2　メソッドが新たなオブジェクトを情報として返す

Graphicsクラスのオブジェクト（変数gが示している）に対して、そのフォントとカラー情報を返してもらい、それぞれ、*myfont*、*mycolor*という名前の変数で指すようにしています。これらの変数は、オブジェクトを参照する変数なので、予めそのクラスの変数として宣言しています。

```
Font    myfont;
Color   mycolor;
myfont = g.getFont( );              // getFontというメソッドを呼び出している
mycolor = g.getColor( );            // getColorというメソッドを呼び出している
```

１１－１－５．１回限りしか使わないオブジェクト

変数で参照しずっと後まで使い続けるオブジェクトもあれば、一回限りの使い捨てのオブジェクトもあります。そのようなオブジェクトは、メソッド呼出しのパラメータの部分に生成するための記述だけが行なわれる場合が多いようです。

```
gc.setColor( new Color( 12, 22, 33 ) );          // カラー設定のためだけに作られたオブジェクト
gc.setFont( new Font( "Serif", Font.BOLD, 12 ) );       // フォント設定のためだけに用いられている
```

上の例は、次のように変数を用意すれば分けて書くこともできます。カラーの方の例だけ示します。

```
Color    temporal;
temporal = new Color( 12, 22, 33 );
gc.setColor( temporal );
```

あるいは、情報を返してくれるメソッドがオブジェクトを返してきてくれるような場合に、そのオブジェクト

が持つ更に別のメソッドを呼び出すような場合は、変数で指さないで、直接ピリオド（.）で区切って呼び出すこともできます。次の例は、変数にカラーオブジェクトを一旦代入し、カラーオブジェクトのメソッドを呼び出す記述です。

```
Color    mycolor = gc.getColor( );
int      red  = mycolor.getRed( );
```

これは、次のように変数を使わない形で記述することが可能です。わかりやすくするために、括弧をつけてありますが、括弧を取っても構いません。

```
int      red  = (gc.getColor( )).getRed( );          // gc.getColor( ).getRed( )と書いてもよい
```

★オブジェクトの消去は？

一旦作られたオブジェクトはどうなるのでしょうか？オブジェクト生成の書式はありますが、削除するための書式はありません。作ってばかりいるとそのうちもう使わない不要なオブジェクトだらけになってしまうのではないでしょうか？

このために、インタープリタ系の言語（Lisp, Smalltalk, Java, Visual Basic AppleScriptなど）では、定期的に不要なオブジェクトを自動的に削除する機能があります。これをガベージ・コレクト（Garbage Collect）と呼んでいます。

変数などで参照されていないオブジェクトは、ガベージ・コレクトの際に自動的に消去されていきます。

１１－１－６．なぜ、アプレットやアプリケーション自身のオブジェクトは作らなくていいの？

アプレットやアプリケーションは、オブジェクトとして生成しなくていいのでしょうか？プログラム中で、**new**Applet()などと記述する必要があるように思えます。プログラムは、クラスを定義しているだけで、そのクラスのインスタンス（オブジェクト）を生成しているようには見えません。

不思議に思えるかも知れませんが、プログラムでは、クラスを定義するだけでいいのです。これには、理由があります。アプレットとアプリケーションで、事情が異なりますので、それぞれの場合について説明しましょう。

アプレットの場合は、ランタイムインタープリタ、たとえばNetscapeやInternet Explorerなどが、クラスの定義を参照して自動的にそのインスタンスを生成するようにしています。ですから、プログラム中でインスタンスを作るように指示しなくてもいいのです。

アプリケーションの場合は、mainという名前のメソッドはクラスメソッドと呼ばれる特殊なメソッドですので、インスタンスを生成しなくても、そのまま実行されてしまうのです。このクラスメソッドについては、後の章で詳しく説明します。

１１－２．AWTのオブジェクトについて

ここでは、AWTのオブジェクトの例として、カラーとフォントの２つのオブジェクトを扱います。

１１－２－１．カラーオブジェクト

Javaのカラーは、RGBモデルを採用しています。RGBモデルでは、色を赤（Red）、緑（Green）、青（Blue）の３つの成分に分け、それを加色混合させています。光の加色混合ですので、よりその成分の色が強くなった場合は、その分だけ明るくなります（注１）。

各色の成分は、0〜255までの整数で強さを調整できます。数が大きい方が、より明るくなります。赤、緑、青と分けて、256段階で調整できるので、256^3（=1677万）色を指定できますが、実際にそれだけの色を表示できるかどうかはディスプレイなどのハードウェアの制約があります。

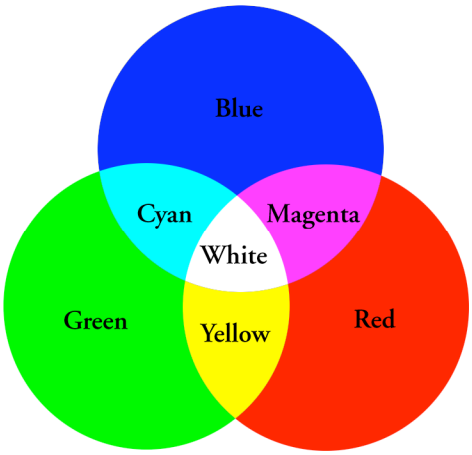

図11-3 RGB関連図

システムで用意されている色について、RGBの値を以下の表に示します。

Name	RGB	Name	RGB
Color.white	255, 255, 255	Color.gray	128, 128, 128
Color.black	0, 0, 0	Color.darkGray	64, 64, 64
Color.lightGray	192, 192, 192		
Color.red	255, 0, 0	Color.magenta	255, 0, 255
Color.green	0, 255, 0	Color.cyan	0, 255, 255
Color.blue	0, 0, 255	Color.pink	255, 175, 175
Color.yellow	255, 255, 0	Color.orange	255, 200, 0

★カラーオブジェクトの生成の仕方
カラーオブジェクトは、次のようなコンストラクタを使って生成することができます。

```
new Color( 赤の強さ, 緑の強さ , 青の強さ )
```

強さは、整数で指定する場合は0〜255の値で指定します。実数で指定する場合は、単精度実数で、0.0f〜1.0fの間で指定します。使いたい色を自分で指定したい場合は、Colorクラスの変数を用意して、次のように、コンストラクタを使ってカラーオブジェクトを生成します。

```
Color  色のための変数  = new Color( 赤の強さ, 緑の強さ , 青の強さ );
```

次の例では、生成したカラーオブジェクトを２つの変数に代入しています。

```
Color   darkred = new Color( 189, 0 , 0 );              // 整数で指定
Color   lightgreen = new Color( 1.0f,  0.2f,  0.2f );   // 単精度実数で指定
```

★透明度を持つカラー
カラーには、透明度を持たせることができるようになりました。この場合は、透明度を表わすパラメータがつきますので、４つパラメータが必要になります。

```
new Color( 赤の強さ, 緑の強さ , 青の強さ , 透明度 )
```

透明度は、整数で指定する場合は0〜255の値で指定します。実数で指定する場合は、単精度実数で、0.0f〜1.0fの間で指定します。ともに、0になると、透明で見えなくなってしまいます。

```
Color   darkred = new Color( 189, 0 , 0, 232 );         // 整数で指定
Color   lightgreen = new Color( 1.0f,  0.2f,  0.2f, 0.1f );   // 単精度実数で指定
```

透明度のあるカラーで図形を描画しますと、その後スライドを重ねるように、重なった図形が透けて見えるようになります。

★描画色の設定の仕方

カラーオブジェクトを描画領域を示すGraphicsクラスのオブジェクトに用意されているsetColorメソッドのパラメータとして指定してやると、それ以降に描画されるものは、指定されたカラーが用いられます。メソッドとしては、既に紹介しましたように次のように用います。

```
setColor( カラーオブジェクト );
```

この２つのメソッドを用いて、次のように描画色を設定することができます。この例では、変数gがGraphicsクラスのオブジェクトを指しています。

```
Color  mycolor = new Color( 140, 140, 200 );      //      カラーオブジェクトを作った
g.setColor( mycolor );                            //      描画色として指定した
```

前にも説明しましたように、一回だけカラー指定するのでしたら、変数で参照しておく必要はありませんから、次のように記述して、使い捨てのオブジェクトにしてしまっても構いません。

```
g.setColor( new Color( 140, 140, 200 ) );         // 色指定のためだけに使われた
```

★現在の色を得る

現在描画色として設定されている色のオブジェクトを得るには、Graphicsクラスに用意されているgetColorメソッドを使います。また、Colorクラスには、RGBのそれぞれの値を取り出すgetRed、getGreen、getBlueメソッドが用意されています。次の例は、現在の描画色のそれぞれの色成分の強さを３つの変数に取り出しています。

```
Color   mycolor = g.getColor( );
int     red = mycolor.getRed( );          // 赤成分の強さを変数redに求める
int     green = mycolor.getGreen( );      // 緑成分の強さを変数greenに求める
int     blue = mycolor.getBlue( );        // 青成分の強さを変数blueに求める
int     alpha = mycolor.getAlpha( );      // 透明度を変数alphaに求める
```

★カラーテーブルを作る例題

例題のプログラムとして、緑を0に固定しながら、16ずつ赤と青の成分を増やしていく、カラーテーブルを作成するようなアプレットを作ってみましょう。緑の値を0以外にしてみたりして、いろいろ試してみなさい。このアプレットの表示には、最低でも幅270、高さ270の描画領域が必要です。変数 blueとredは、それぞれ色の成分の強さを表していますが、同時にカラーテーブルのそれぞれの四角形のx座標、y座標としても使われています。

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

public void paint( Graphics g ) {
        for ( int blue=0; blue<256 ; blue += 16 ) {
                for ( int red=0 ; red < 256 ; red += 16 ) {
                        Color mycolor = new Color( red, 0, blue );
                        g.setColor( mycolor );
                        g.fill3DRect( blue+30, red+30, 14, 14, true );
                }
        }
}
```

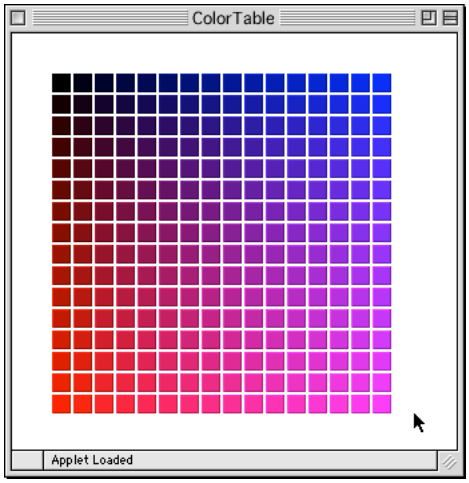

図11-4 アプレットの実行例

注１　印刷では、減色混合になります。減色混合とは、色を足していけばいくほど、黒に近づいていくものです。絵の具で使われる色を混ぜ合わせていったときのことを思い出してください。そのために、RGBモデルの替わりに、CyanとMagenta、Yellow、および黒の微妙な色調整のためのblacKを用いたCMYKモデルで色を指定しています。これらの３色とRGBの３色の関係は、RGB関連図をみてください

１１－２－２．フォントについて

フォント（Font）は、タイプフェース（Typeface）とも呼ばれることがありますが（注１）、日本語では文字の字体のことを表しています。文字をグラフィックスとして描画するときに、使われる文字のデザインのことを指しています。Javaのアプレットでは、描画領域上にさまざまなフォントを持つ文字列を表示させることができます。ここでは、フォントを例に取ってオブジェクトの使い方を学んでいきます。

★フォントオブジェクトの生成の仕方

フォントオブジェクトは、次のようなコンストラクタを呼び出すことによって生成させることができます。

```
new Font( フォント名, スタイル, サイズ )
```

このコンストラクタのパラメータについて説明する前に、もう少しフォントについて説明しましょう。古くはXeroxのAltoコンピュータやMacintoshから、最近ではUnix上のウィンドウシステムやWindowsなども、文字を特定のフォントで表示できるようになっています。コンピュータシステム上では、フォントは、フォントファミリー（Font Family）と呼ばれるいろいろなスタイル、いろいろな大きさのサイズのフォントの集まりとして管理されています。

１つのフォントのデザインから、様々なバリエーションを持つ字体が作られるようになってきました（注２）。そのために、フォントオブジェクトを生成するときに、基本となるフォントの名前とスタイルとサイズを指定するのです。

フォント名は、文字列として指定します。使えるフォントの名前は、JDK1.0とJDK1.1では異なります。ブラウザなどによっては、JDK1.1の名前に対応していないものもあるかも知れませんので、ここで代表的な３つのフォントについて、両方の名前を載せておきます。また、Java2は、基本的にはJDK1.1のものは踏襲しています（注３）。

▼使えるフォントの名前

フォント名（JDK1.0）	フォント名（JDK1.1）	字体の目安
TimesRoman	Serif	ABCDEFGHIJKLMNabcdefghijklmn012345
Helvetica	SansSerif	ABCDEFGHIJKLMNabcdefghijklmn012345
Courier	MonoSpaced	ABCDEFGHIJKLMNabcdefghijklmn012345

JDK1.0のフォント名は、通常に世間で使われている名前になっています。JDK1.1のものは、フォント名というよりも、フォントの分類を表現している名前です。フォントの分類について少し説明しますと、TimesRomanというフォントは、セリフフォントとして分類されています。Helveticaはサンセリフフォント、Courierはスラブセリフフォントとしてそれぞれ分類されています。ここで、セリフ（Serif）というものについて知っておく必要があります。

図11-5　セリフの図

セリフとは、上の図のようにローマ字を表現している縦棒や横棒の端に付けられている装飾のことです。これは、元はローマ時代に石碑に字を刻印するときに使われた鑿（のみ）の跡です。これが後世になっても装飾として使われるようになったものです。セリフがあるのがセリフフォント（注４）、セリフがないのがサンスセリフ（Sansはwithoutの意味です）と呼ばれています。Courierはセリフフォントの一種のスラブセリフフォントに属しています。スラブセリフフォントの特徴は、セリフの部分が単純な水平の横棒、垂直な縦棒になっていることです。

しかし、Courierには、JDK1.1ではMonoSpacedという名前がついています。これは、Courierが固定幅のフォントであることを示しています。次の図のように文字ごとに幅が異なるフォントのことを可変幅（Propotional）フォントと呼んでいます。一方、どの文字でも幅が同じフォントのことを等幅あるいは固定幅（Fixed）フォントと呼んでいます。MonoSpacedは、固定幅フォントのことを意味しています。

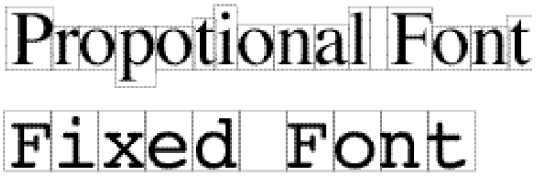

図11-6 可変幅フォントと固定幅フォント

他にも様々なフォントがあるのですが、どのシステムでも使えるこの３種類のフォントを使うことをお勧めします。プログラムを作成したコンピュータでは当然のように使えるフォントでも、アプレットとしてネットワークを介して移動した先のコンピュータシステムでは、インストールされていないかも知れません。なお、これ以外に、Dialog、DialogInput、Symbolといったフォント名もJavaでは定義しています（注５）。

スタイルとしては、通常の字体以外に、太字や斜体（注６）も次のように指定することができます。

▼指定できるスタイル

スタイル名	内容	スタイルの目安
Font.PLAIN	通常の字体	Typography
Font.BOLD	太字体	**Typography**
Font.ITALIC	斜体	*Typography*
Font.BOLD+Font.ITALIC	太字斜体	***Typography***

サイズは、文字のおおよその大きさを示すものでポイントで指定します。１ポイントは、1/72インチ（0.35mm）ですが、これは大きさの目安だと思ってください。大きさを可変にできるフォントのことをスケーラブル（Scalable）フォントと呼びますが、コンピュータシステムによってはスケーラブルフォントではない場合もありますので、きちんと表示されるかどうかは注意してください。サイズとしては、システムによって異なりますが、最低４ポイントぐらいから最高128ポイントぐらいまでと考えた方がいいでしょう。

▼サイズ

ポイント（整数）で指定

図11-7　サイズの違う文字

★フォントの設定の仕方

それ以降描画する文字のフォントを設定するのには、カラーオブジェクトと同じように、Graphicsクラスのオブジェクトには、setFontというメソッドがあります。これを利用します。

```
g.setFont( フォント );
```

たとえば、次のようにフォントを指定することができます。

```
Font  myfont = new Font( "Serif", Font.ITALIC,  24 );
g.setFont(  myfont  );
```

このような指定も、次のように１行で書けます。次のプログラムの断片では、更に指定されたフォントを用いて何かメッセージを表示させています。

```
g.setFont(  new  Font( "Serif",  Font.ITALIC,  24 ) );
g.drawString( "A big message for you", 10, 100 );
```

★現在のフォントを知る方法

Graphicsクラスのオブジェクトには、getFontメソッドがあり、これを用いて現在のフォント（を示すオブジェクト）を知ることができます。また、Fontクラスのオブジェクトには、getName、getStyle、getSizeといったメソッドが用意されています。次のプログラムの断片は、現在のフォントの情報をそれぞれの変数に求めています。

```
Font    myfont = g.getFont( );
String  fontname  = myfont.getName( );      // 現在のフォントの名前を求める
int     style = myfont.getStyle( );         //  現在のフォントのスタイルを求める
int     size = myfont.getSize( );           //  現在のフォントのサイズを求める
```

★フォント情報オブジェクトの生成の仕方と利用法

文字の実際の幅や高さを求めるには、FontMetricsクラスが用いられます。このクラスのオブジェクトは、特定のフォントについての情報を持っています。

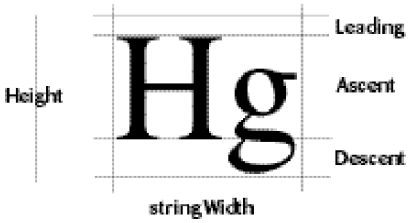

図11-8 フォントの幅と高さについて

フォントの高さには、図のようにいろいろな部分があります。基底ライン（Baseline）があり、そこからの文字の高さ（Ascent）と、基底ラインのから下にでている部分（Descent）があり、両者を足したものが高さ（Height）になっています。また、上部の余白の部分（Leading）も設けられています。

FontMetricsクラスのオブジェクトには、それぞれの高さや幅を求めるための次のようなメソッド群が用意されています。

getAscent( )	基底ラインからの高さを求める
getDescent( )	基底ラインの下にはみ出ている部分の高さを求める
getHeight( )	高さを求める
getLeading( )	上部の余白の高さを求める
stringWidth(文字列)	指定された文字列の幅を求める

それぞれの高さや幅は、ドット（ピクセル）単位で求められます。すなわち、整数値として返されます。さて、FontMetricsのオブジェクトはどうやって求めるのでしょうか？AppletやGraphicsクラスのオブジェクトにはgetFontMetricsという名前のメソッドが用意されています。このメソッドは、アプレットなどの実行環境におけるフォント情報を持つオブジェクトを返してくれます。getFontMetricsメソッドは、次のような２種類が用意されており、パラメータとしてフォントを指定しない場合は、現在のフォントの情報を求めることになります。なお、前者はAppletクラスでは使えません。

getFontMetrics( )	現在設定されているフォントの情報を求める （Graphicsクラスのオブジェクトのメソッド）
getFontMetrics(フォント)	指定したフォントの情報を求める

いくつかの例を記述してみましょう。現在設定されているフォントを使ってMessageと画面に表示する場合に、その文字列がどれくらいの幅であるかを変数widthに求めてみましょう。

```
FontMetrics     metrics = g.getFontMetrics( );
int             width = metrics.stringWidth(  "Message" );
```

上の例では、Graphicsクラスのオブジェクトを指す変数gcを使ってFontMetricsクラスのオブジェクトを求めていましたが、次のように直接アプレットの方のgetFontMetricsメソッドを利用することもできます。その場合は、変数を指定する必要はありません。次の例では、12ポイントの太字のSansSerif（Helvetica）フォントの高さを変数heightに求めています。

```
Font            myfont = new Font( "SansSerif", Font.BOLD, 12 );
FontMetrics     metrics = getFontMetrics( myfont );
int             height  = metrics.getHeight( );
```

前に説明しましたように、これ以降フォント情報を参照しない場合は、同じことを変数を使わないで次のように書くことができます。わかりやすいように括弧をつけて書いておきました。

```
Font  myfont = new Font( "SansSerif", Font.BOLD, 12 );
int   height  =  (getFontMetrics( myfont )).getHeight( );
```

注１　歴史的には、FontとTypefaceは区別されていました。特定のサイズ、スタイルで用いられている字体のことをFontと呼び、字体のデザインをTypefaceと区別して呼ぶこともあります。

注２　AdobeType1フォントなどでは、厳密な美しい字体を表現するために、スタイルごとに別のフォントとして定義されています。
注３　Java2では、更にAdobe Type1フォントなどが扱えるので、フォント用の別パッケージがクラスライブラリィに用意されています。システムに依存したフォントや各国語対応のフォントの扱いができるのですが、ここでは無難なJDK 1.0あるいはJDK 1.1ベースのフォントを使うことにします。NetscapeやInternet Explorerなどがどこまで対応するのか原稿を書いている時点ではわからないからです。
注４　セリフフォントは更に細かく、オールドスタイルフォント、モダンフォント、スラブセリフフォントに分類することができます。
注５　Javaではフォントを設定しない場合は、たぶんDialogフォントが使われます。このフォントに対して、実際にはどのようフォントが使われるのかに関しては、システムによって異なります。
注６　フォントをデザインとして扱うタイポグラフィ（Typography）の世界では、セリフフォントの場合の斜体はイタリック体と呼んでいますが、サンスセリフフォントの場合は、斜体をオブリーク（Oblique）体と呼んでいます。

１１－３．日付・時刻について

１１－３－１．Calendarクラスと日時

java.utilパッケージの中のCalendarクラスは、日付や時間を保持するための基本的なクラスになっています。このクラスのオブジェクトを利用すると現在の日付や時間などを求めることができます。

★現在の時間を求めたいとき

次の代入文は、Calendarクラスのオブジェクトを指す変数*cal*を宣言し、それに次のような代入を行ないます。

```
Calendar   cal = Calendar.getInstance( );
```

ところが、これではグリニッジ標準時間を計算してしまいます。これは困りますね。英国の時間から日本の時間に直すのって、日付も変わりますから結構大変な作業です。そこで、次のようにグレゴリアカレンダーで時差を考えて、日本の時間に直すことができます。

```
Calendar   cal = new GregorianCalendar( TimeZone.getTimeZone("JST") );
```

GregorianCalendarクラスは、Calendarクラスのサブクラスになっています。このグレゴリアカレンダーは、現在世界の標準のカレンダーになっていて、紀元前（BC）4,716年から、紀元後（AD）5,000,000年まで表せるものです。また、内部的には協定世界時間（UTC）に基づいて計算されています。

１１－３－２．TimeZoneクラスによる時間帯

時差を伴った時間帯を表すためにTimeZoneクラスが用意されています。このクラスに用意されてるメソッドgetTimeZoneを使って、日時を求めるときに地球上の各時間帯の時刻に直すことができます。この時間帯は、経度だけでなく、国によっても違いますので、文字列でその時間帯を指定しています。JSTは、Japan Standard Time（日本標準時間）の頭文字になっています。これ以外に次のような時間帯があります。Java1.1〜Java2で使える代表的なものについて正式名称と時差と共に挙げてみました。

GMT	Greenwich Mean Time	UTC	Universal Time Coordinated (GMT)
ECT	Europe Central Time (GMT+01)	EET	Europe Eastern Time (GMT+02)
ART	Arabic Egypt Standard Time (GMT+02)	EAT	Eastern African Time (GMT+03)
MET	Middle East Time (GMT+03:30)	NET	Near East Time (GMT+04)
PLT	Pakistan Lahore Time (GMT+05)	IST	India Standard Time (GMT+05:30)
BST	Bangladesh Standard Time (GMT+06)	VST	Vietnam Standard Time (GMT+07)
CTT	China Taiwan Time (GMT+08)	JST	Japan Standard Time (GMT+09)
ACT	Australia Central Time (GMT+09:30)	AET	Australia Eastern Time (GMT+10)
SST	Solomon Standard Time (GMT+11)	NST	New Zealand Standard Time (GMT+12)
MIT	Midway Islands Time (GMT-11)	HST	Hawaiian Standard Time (GMT-10)

AST	Alaska Standard Time (GMT-09)	PST	Pacific Standard Time (GMT-08)
PNT	Phoenix Standard Time (GMT-07)	MST	Mountain Standard Time (GMT-07)
CST	Central Standard Time (GMT-06)	EST	Eastern Standard Time (GMT-05)
IET	Indiana Eastern Standard Time (GMT-05)	PRT	Puerto Rico Time (GMT-04)
CNT	Canada Newfoundland Time (GMT-03:30)	AGT	Argentina Standard Time (GMT-03)
BET	Brazil Eastern Time (GMT-03)	CAT	Central African Time (GMT-1)

これらの３文字の略称は、同じ名前が一杯ありますので、Javaの次の版（JDK1.3）ではサポートされるものの、推奨されないようです。もし、その国の標準時間の名称がわからなければ、単にGMT（グリニッジ標準時間）に対しての差分で指定することができます。以下の書式で、‖は省略可能であることを示します。また、｜はどちらかということを示しています。時間はhhの２桁で、また分指定もmmの２桁で指定します。

GMT[+｜-]hh[:mm]　例えば、GMT12　GMT-04　GMT+09:00　など

１１－３－３．日付・時刻を得たいとき

Calendarクラスのオブジェクトさえ求められれば、そのオブジェクトが求められた時点での日付や時刻を簡単に得ることができます。以下の例は、代表的なものについて変数の宣言をしながら実際に求めてみたものです。これらは、先ほど求めた変数*cal*が指すオブジェクトのメソッドgetを呼び出して、求めています。

```
int year = cal.get( Calendar.YEAR );                          //  西暦の年
int month = cal.get( Calendar.MONTH ) + 1;          //  月（0〜11が返ってくるので+1する）
int weekofyear = cal.get( Calendar.WEEK_OF_YEAR );          //  その年の何週目か
int weekofmonth = cal.get( Calendar.WEEK_OF_MONTH );    //  その月の何週目（1〜）
int dayofyear = cal.get( Calendar.DAY_OF_YEAR );            //  その年の何日目（1〜）
int day  = cal.get( Calendar.DATE );                          //  その月の何日目（1〜）
int dayofweek = cal.get( Calendar.DAY_OF_WEEK ); //  その週の何日目（日曜の1〜7）
int ampm  = cal.get( Calendar.AM_PM );                        //  午前午後（午前が0、午後が1）
int hour = cal.get(  Calendar.HOUR );                         //  何時（0〜12時）
int dayhour = cal.get(  Calendar.HOUR_OF_DAY );  //  何時（0〜23時）
int minute = cal.get(  Calendar.MINUTE );                     //  何分（0〜59分）
int second  = cal.get(  Calendar.SECOND  );         //  何秒（0〜59秒）
int millisecond  = cal.get(  Calendar.MILLISECOND  );         //  ミリ秒（0〜999秒）
```

なお、Calendar.DATEは、Calendar.DAY_OF_MONTHとも記述できます。それでは、現在の年月日と曜日、時刻を表示するようなアプリケーションプログラムを記述してみましょう。Calendarクラスは、java.utilパッケージなので、**import**の行を忘れずに記述してください。

```
import  java.util.*;                    // java.utilパッケージのクラスを使う

public class  CurrentDisplay {
        public static void  main(  String [ ] args ) {
                Calendar   cal = new GregorianCalendar( TimeZone.getTimeZone("JST") );
                System.out.println( "Year: " + cal.get( Calendar.YEAR ) + "  Month: " +
                        ( cal.get( Calendar.MONTH )+1 ) + "  Day:  " + cal.get( Calendar.DATE ) );
                int  weekday =  cal.get( Calendar.DAY_OF_WEEK ) ;
                if ( weekday == 1 ) { System.out.println( "Sunday" ); }
                else if ( weekday == 2 ) { System.out.println( "Monday" ); }
                else if ( weekday == 3 ) { System.out.println( "Tuesday" ); }
                else if ( weekday == 4 ) { System.out.println( "Wednesday" ); }
                else if ( weekday == 5 ) { System.out.println( "Thursday" ); }
                else if ( weekday == 6 ) { System.out.println( "Friday" ); }
                else  { System.out.println( "Saturday" ); }
                if  ( cal.get( Calendar.AM_PM ) == 0 ) { System.out.print( "AM " ); }
                else   { System.out.print( "PM " ); }
                System.out.println( cal.get( Calendar.HOUR ) + ":" + cal.get( Calendar.MINUTE )
```

```
                                        + ":" + cal.get( Calendar.SECOND ) );
                }
        }
```

１１－３－４．日時、時刻を指定する

Calendarクラスのオブジェクトに日時を指定するには、日時をすべてクリアするclearメソッドと、次のような３つの種類のsetメソッドが用意されています。注意しなければならないのは、月の指定は０から始まることです。ですから、月には一つ小さい値を設定します。何時かは２４時間（0〜23）で指定します。

```
set( 年, 月 , 日 )
set( 年, 月 , 日 , 時, 分 )
set( 年, 月 , 日 , 時, 分, 秒 )
```

次のプログラムの断片は、上記の３つのメソッドを使って、日時を指定しているものです。月が9になっていますが、月の指定は0から始まりますので、10月の指定になっています。

```
Calendar   cal = new GregorianCalendar( TimeZone.getTimeZone("JST") );
cal.clear( );                                   // 日時をクリアする
cal.set( 2000, 9, 1 );                          // 2000年10月1日に指定
cal.set( 2000, 9, 1, 14, 12 );                  // 2000年10月1日14時12分に指定
cal.set( 2000, 9, 1, 14, 12, 45 );              // 2000年10月1日14時12分45秒に指定
```

１１－３－５．時間差とDateクラス

現在の時間が一定の時刻からどの程度過ぎたかを知りたい場合があります。この場合には、Dateクラスという Calendarクラスの前に時間差を考慮しないで作られたクラスを利用します。このクラスもjava.utilパッケージの中に入っています。Calendarクラスのオブジェクトから Dataクラスのオブジェクトを求めるgetTimeメソッドが用意されていますので、次のような形で求めることができます。

```
Date     caldate = cal.getTime( );
```

Dataクラスのオブジェクトには、toStringメソッドが用意されており、設定されている時間帯での標準的な日時表示の文字列を返してくれます。

```
System.out.println( caldate.toString( ) );
```

なお、System.out.printlnは、パラメータがオブジェクトの変数であった場合は、そのオブジェクトの toStringメソッドを自動的に呼び出すような仕組みになっていますので、以下のように記述しても構いません。

```
System.out.println( caldate );
```

Dataクラスのオブジェクトには、toStringメソッド以外に、1970年の１月１日０時０分０秒からの経過時間を求める同じ名前のgetTimeメソッドがあります。経過時間は、型は**long**の整数型でミリ秒で求められます。下の代入文では、Calendarクラスのオブジェクトを指す変数に対して更にgetTimeメソッドを呼び出して、Date クラスのオブジェクトを求め、そしてそのオブジェクトに対しておgetTimeメソッドを呼び出して、経過時間を求めています。２つの時刻に対して、この方法を使って経過時間を求め、その差分を出せば、どの程度時刻が過ぎたかわかります。また、求められた時間差に対して、1000で割ると秒数が、更にその結果を60で割ると分数（更にその結果を60で割ると時間）を得ることができます。

```
long     millisecond = cal.getTime( ).getTime( );        // Dateクラスを求め、そのgetTimeを呼び出す
```

これを２行に分けて記述しますと以下のように記述できます。

```
Date     caldate = cal.getTime( );
long     millisecond = caldate.getTime( );
```

次のアプリケーションプログラムは、起動してから１秒ごとに表示をして、６０秒経つと終了するものです。開始時間に対する経過ミリ秒を変数*start*に求めています。また、刻々と変わる時間の経過ミリ秒を変数*now*に求めています。この２つの変数の差を1000で割った時間差（秒）を変数*second*に計算しています。この例題は、**do～while**文を使った方がすっきりと記述することができます。このように、一度やってみて、後から繰返しするかどうかを判定したい場合は、**do～while**文の方が同じことを何回も書かなくて済みます。

```
import   java.util.*;
public class  WaitMinute {
        public static void  main(  String [ ] args ) {
            Calendar   origin = new GregorianCalendar( TimeZone.getTimeZone("JST") );
            long start = origin.getTime( ).getTime( );
            long second =  0;


            do {
                Calendar  current = new GregorianCalendar( TimeZone.getTimeZone("JST") );
                long now = current.getTime( ).getTime( );
                if ( (now - start) / 1000 > second ) {              // 次の秒になったら
                        second = (now - start)/1000;                // 時間差を秒に変換
                        System.out.println( second + " seconds" );
                }
                second = (now - start)/1000;
            } while ( second < 60 ) ;
}
```

このプログラムは、ずっと繰返しを行ないながら時間を待っている（Active Waiting）ので、コンピュータを無駄に稼働させています。このプログラムを実行させると、遅いコンピュータでも毎秒少なくとも100,000回以上は無駄に繰返しをしています。実際のプログラム製作ではこれは避けた方がいいでしょう。コンピュータに別の計算をするように解放しておいてから所定の時間を待つ方法は、後の章で出てきます。

１１－４．課題

11-1.
カラー配列の例題のアプレットプログラムは、横方向に青色、縦方向に赤色の色成分を変化させていましたが、横方向に赤色、縦方向に緑色の色成分を変化させてみなさい。青色成分は、適当な定数値（0～255）を選びなさい。

11-2.
フォントのサイズやスタイルを変えながら（たとえば9ポイントから36ポイントぐらいまでの適当なポイント数を複数選びなさい）、同一のメッセージを表示するようなアプレットプログラムを作りなさい。クラス名は、FontListingにしなさい。

11-3.
フォントを変えながら、横に表示するようなアプレットプログラムを作りなさい。フォントのスタイルは、通常のスタイル（変えても構いません）で、サイズは18ポイント（変えても構いません）とします。クラス名は、HetroFontにしなさい。

```
This is a TimesRoman,  and  Helvetica  and Courier.
```

11-4.
時間差を求める例題のプログラムを参考にしながら、１秒毎に現在の日付（年／月／日）と時間（時：分：秒）を表示するプログラムを作りなさい。時間は24時間表記にしなさい。なお、30秒表示したら終了するようにしなさい。クラス名は、CurrentTimeにしなさい。