NAL

クラスライブラリ
リファレンス

版	日付
1.0	1999/08/05
2.0	2002/04/09
3.0	2003/02/19

はじめに

知的財産権について

アクト株式会社

改訂履歴

2007/01/21	class.Parser を追加
2009/10/26	class SMTP に port 指定を追加

目次

1 全角かな文字を扱うクラス

```
import extends.String.Kana

extends String
```

コンストラクタ	
かな(str:string)	例：new かな("へのへのもへじ１２３")

クラス・メソッド	

インスタンス・メソッド	
全角 to 半角(spec:string="ア"):string	保有値のうち、spec に指定する文字種に従い、全角かなと全角カナを半角ｶﾅに、全角数字を半角数字に、全角英字を半角英字に変換した結果を返す。 spec 文字列中に"ア"が含まれていれば全角カナを、"あ"が含まれていれば全角かなを、"Ａ"または"A"が含まれていれば全角英字を、"０"または"0"が含まれていれば全角数字を、それぞれ半角に変換する。該当しない文字は変換しない。保有値には影響を与えない。
かな to カナ():string	保有値のうち、全角かなを全角カナに変換して保有し、変換した結果の文字列を返す。 該当しない文字は変換しない。
カナ to かな():string	保有値のうち、全角カナを全角かなに変換して保有し、変換した結果の文字列を返す。 "ヴ" は "ぶ"に変換する。該当しない文字は変換しない。

2 行列を扱うクラス

```
import class.Matrix
class Matrix.Dimension(row:int,clm:int);
```

コンストラクタ	
Matrix(elements[]:Array={})	elements に指定する m 行 n 列の数値配列を要素とする行列オブジェクトを生成する

インスタンス・メソッド	
get(row:int,clm:int):number	row 行 clm 列の要素を得る
put(row:int,clm:int,value:number)	row 行 clm 列の要素に value を設定する
setRow(row:int,elements:number[])	row 行に列要素配列を設定する
setClm(clm:int,elements:number[])	clm 列に行要素を設定する
setElement(elements:Array)	行列[row][clm] を設定する
add(adder:Matrix):Matrix	adder 行列を加算する
sub(adder:Matrix):Matrix	adder 行列を減算する
mul(multiplier:number):Matrix	スカラー積
mul(multiplier:Matrix):Matrix	行列積
getDimension():Dimension	行数と列数を得る
setDimension(row:int,clm:int)	行数と列数を設定する

インスタンス・オペレータ	
+ (value:Matrix):Matrix	add(value:Matrix)に同じ
- (value:Matrix):Matrix	sub(value:Matrix)に同じ
* (value:number):Matrix	mul(value:number)に同じ
* (value:Matrix):Matrix	mul(value:Matrix)に同じ

■使用例：

```
import class.Matrix;
var     m1_elements = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};
var     m2_elements = {{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15},{16,17,18,19,20}};
var     m1:Matrix = Matrix.new(m1_elements);
var     m2:Matrix = new Matrix(m2_elements);
var     m3:Matrix = m1 * m2;
```

3 分数を扱うクラス

```
import class.Fraction
```

コンストラクタ	
Fraction(numerator:int=0, denominator:int=1)	numerator を分子、denominator を分母とする分数オブジェクトを生成する

インスタンス・メソッド	
reduce():Fraction	約分し、自身の参照値を返す
add(adder:Fraction):Fraction	adder で指定する分数を加算し、自身の参照値を返す (obj += adder)
sub(adder:Fraction):Fraction	adder で指定する分数を減算し、自身の参照値を返す (obj -= adder)
mul(adder:Fraction):Fraction	adder で指定する分数を乗算し、自身の参照値を返す (obj *= adder)
div(adder:Fraction):Fraction	adder で指定する分数で除算し、自身の参照値を返す (obj /= adder)
toReal():real	実数化した結果を返す
toString():string	"分子/分母"書式に文字列化して返す

インスタンス・オペレータ	
+= (value:Flaction):Flaction	add(value:Flaction)に同じ
-= (value:Flaction):Flaction	sub(value:Flaction)に同じ
*= (value:Flaction):Flaction	mul(value:Flaction)に同じ
/= (value:Flaction):Flaction	div(value:Flaction)に同じ
== (value:Fraction):bool	toReal()と value.toReal()が等しければ true
!= (value:Flaction):bool	toReal()と value.toReal()が等しくなければ true

■使用例：

```
import class.Fraction;
var     val = new Fraction(1,3);                    1/3
val.add(new Fraction(1,4)).toString();              1/3+1/4 --> 7/12
val.mul(new Fraction(3)).toString();                7/12 * 3 --> 7/4
```

4 XML を扱うクラス

```
import class.XML
```

コンストラクタ	
XML()	

クラス・メソッド	
parseXML(xmldoc:Buffer):XML	xmldoc に与えられた shift-JIS コードで記述された整形式 XML データを DOM 解析した結果の XML オブジェクトを得る

インスタンス・メソッド	
getName():string	ノードのタグ名を得る
attributeMember():string[]	ノード内の属性名配列を得る
elementMember():string[]	ノードの子ノード名の配列を得る。子ノードがない場合は 空の配列を返す
elementLength(key:string):int	key で指定する同名の子ノード数を得る。 指定する子ノードがない場合は 0 を、子ノードがあれば 1 以上の値を返す。 2 以上が返された場合は、そのノードの指定には配列要素番号を指定しなければならない。
elementType(key:string):string	key で指定する要素がノードの場合は"NODE"、同名の要素が複数ある場合は"ARRAY"、該当する要素がない場合は"NONE"を返す
nodeType():string	ノードが要素を持たない場合は"EMPTY"、文字列値を持つ場合は"#PCDATA"、子ノードを持つ場合は"COMPLEX"を返す
getValue():string	ノードが文字列要素を持つ場合その値を得る。
getValue(path:string):string	path で指定するノード値あるいは属性値を得る。返される値は getValue()に順ずる。path は直下の単独のノード名あるいは属性名か、各々のノード名をもしくはノード名と属性名を"/"で区切ったものであり、最終値に属性名を指定した場合には属性値を得る。該当するノードあるいは属性がない場合は null を返す。属性名は@で先導しなければならない。 path が"/"で始まる場合は ROOT ノードからの絶対パスとなり、".."は親ノードを意味する。 path 中に配列ノードがあり、その要素番号を指定していない場合は先頭要素を指定したとみなす
getAttribute():Attribute	ノードの属性オブジェクトを得る。属性がなければ null を返す。 属性オブジェクトは object Attribute 型の静的オブジェクトである
getAttribute(key:string):string	ノード内の、key で指定する属性値を得る ※getValue("@"+key)と同じ

node(path:string):XML	path で指定するノードを得る。path は直下の単独のノード名か各々のノード名を"/"で区切ったもの。該当するノードがない場合は null を返す。 path が"/"で始まる場合は ROOT ノードからの絶対パスとなり、".."は親ノードを意味する。 path 中に要素番号を付けない配列ノードがある場合は先頭要素を指定したものとみなす
roots():XML	ROOT ノードを返す。※node("/")と同じ
child():XML	先頭の子ノードを得る。なければ null を返す
parent():XML	親ノードを得る。なければ null を返す。※node("..")と同じ
next():XML	同一階層の次のノードを得る。なければ null を返す
setName(name:string)	ノードのタグ名として name を設定する
setValue(val:string)	val をノードの要素値として設定する
addValue(val:string)	要素値に val を追加する
setAttribute(key:string, val:string)	ノード内の、key で示す属性に val を設定する
addChild(node:XML)	node を子ノードとして追加する

■使用例：

変数 xmldata に、以下の整形式 XML データを Buffer オブジェクトとして格納しているとする。

```
<?xml version="1.0" encoding="Shift-JIS" ?>
<発注伝票 発注日="2001-10-14">
  <発注者 会社名="アクト株式会社" 住所="文京区千石 1-1-16" 電話番号="03-5547-6287" />
  <注文>
    <商品 品名="ACTOS" 個数="20" 単価="10000" 金額="200000" />
    <商品 品名="NAL" 個数="1" 単価="800000" 金額="800000" />
    <商品 品名="GUIDE" 個数="2" 単価="400000" 金額="800000" />
    <小計 品目数="23">1800000</小計>
    <消費税>90000</消費税>
    <合計金額>1890000</合計金額>
  </注文>
</発注伝票>
```

これをオブジェクトとして解析する parseXML()メソッドの結果に対し、各々のメソッドが何を返すかを以下に示す。

	オペレーション	解説
	var xroot:XML = parseXML(xmldata)	xroot に発注伝票ノード（ROOT）のオブジェクトが得られる
	xroot.getName()	"発注伝票" を返す

	xroot.elementMember()	{"発注者","注文"} を返す
	xroot.attributeMember()	{"発注日"} を返す
	xroot.getValue("@発注日")	発注日属性の値 "2001-10-14" を返す
	xroot.getValue("発注者/@会社名")	発注者ノードの会社名属性の値 "アクト株式会社" を返す
	var node:XML = xroot.node("発注者")	node に発注者ノードのオブジェクトが得られる
	node.elementMember()	発注者ノードは要素を持たないため、{} を返す
	node.attributeMember()	{"会社名","住所","電話番号"} を返す
	node.getAttribute("会社名")	会社名属性の値 "アクト株式会社" を返す
	node.getValue("@電話番号")	電話番号属性の値 "03-5547-6287" を返す
	var order:XML = xroot.node("注文")	order に注文ノードのオブジェクトが得られる
	order.elementMember()	{"商品[0]","商品[1]","商品[2]","小計","消費税","合計金額"} を返す
	order.attributeMember()	注文ノードは属性を持たないため、{} を返す
	order.elementType("商品")	注文ノードの商品要素は同名要素があるため "ARRAY" を返す
	order.elementType("商品[1]")	注文ノードの商品[1]要素はノードであるため "NODE" を返す
	order.elementLength("商品")	注文ノードの商品要素は同名要素が 3 個あるため 3 を返す
	order.getValue("小計")	小計要素の値 "1800000" を返す
	order.getValue("小計/@品目数")	小計要素の品目数属性の値 "23" を返す
	order.nodeType()	注文ノードは子ノードを持つため "COMPLEX" を返す
	var item:XML = xroot.node("注文/商品")	item に商品[0]ノードのオブジェクトが得られる
	item.attributeMember()	{"品名","個数","単価","金額"} を返す
	item.getValue("@品名")	品名属性の値、"ACTOS" を返す
	var item_n:XML = item.next()	item_n に商品[1]ノードのオブジェクトが得られる
	item_n.getValue("@品名")	品名属性の値、"NAL" を返す

■XML データの文字エンコーディングについて
XML クラスは XML データの文字列をシフトＪＩＳのみとしている。
他の文字コードで記述されたデータについては事前にシフトＪＩＳに変換した後に parseXML()をおこなわなければならない。
例：xmldata.xml というファイルに UTF-8 コード体系の XML データが格納されている場合

```
var     utf8_data:Buffer = new Storage("xmldata.xml").get();            // ファイルから読み込み
var     shift_jis_text:string = utf8_data.getUTF8Text();                // utf-8 データを shift-JIS 文字列に変換
var     shift_jis_data:Buffer = new Buffer().putString(shift_jis_text); // shift-JIS 文字列を Buffer オブジェクトに変換
var     xmlobj:XML = XML.parseXML(shift_jis_data);                      // XML を DOM 解析
```

のようにする。

5 List クラス

```
import class.List
```

抽象クラス

コンストラクタ	

クラス・メソッド	
getTop():List	最近生成したオブジェクトへの参照値を返す

インスタンス・メソッド	
getNext():List	直前に生成されたオブジェクトへの参照値を返す。なければ null
remove()	リストから削除

※List クラスは includes して使用すること

■使用例：

```
import class.List;
class   UserList(name:string) includes List{
        public  method  view()  writeln("Name=",name)
        }

new UserList("花子");
new UserList("太郎");

var     top:List = UserList.getTop();   // 最近生成したオブジェクト
while(top instanceof UserList){
        top.view();
        top = top.getNext();            // 過去に生成したオブジェクト
        }
```

6 唯一のインスタンスを生成するクラス

```
import class.Singleton
```

抽象クラス

コンストラクタ	

クラス・メソッド	
getInstance(arglist:variant) :Singleton	最近生成したオブジェクトへの参照値を返す

※Singleton クラスは includes して使用すること

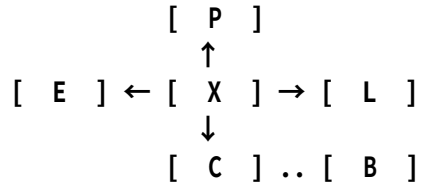
■使用例：

```
import class.Singleton;
class   UserClass(argumentsList) includes Singleton{ ... }

var     inst_1:UserClass = UserClass.getInstance();
var     inst_2:UserClass = UserClass.getInstance();
if(inst_1 == inst_2)    writeln("同じインスタンスです");
```

あるクラスのインスタンスをただ１つしか生成したくない場合には、そのクラスに Singleton **クラスを** include **し、インスタンスの生成には** new **によるクラス・コンストラクタを使用しないで** getInstance()**メソッドを利用する。**
getInstance()**メソッドは、インスタンスが既にあるかないかを調査し、初めて作成する場合はクラスのコンストラクタを呼び出し、生成されたインスタンスを結果として返す。**
既にインスタンスがある場合は、新たなインスタンスを生成することなく、既存インスタンスへの参照値を返す。
getInstance()**メソッドに引き渡した実引数は、新たなインスタンスを生成する場合にのみコンストラクタ引数として渡される。**

7 相互関係を扱うクラス（抽象クラス版）



で示す、多階層にわたる相互関係を扱う。デストラクタによって親が消滅するタイミングでその全ての子が消滅する。

```
import class.PCBEL
```

抽象クラス

インスタンス・メソッド	
insert(obj:PCBEL):PCBEL	obj を兄として挿入する
append(obj:PCBEL):PCBEL	obj を弟として接続する
inschild(obj:PCBEL):PCBEL	obj を長男として接続する
addchild(obj:PCBEL):PCBEL	obj を末子として接続する
isparent(obj:PCBEL):PCBEL	obj が自身の祖先なら obj を返し、そうでないなら null を返す
ischild(obj:PCBEL):PCBEL	obj が自身の子孫なら obj を返し、そうでないなら null を返す
parent():PCBEL	親を返す
child():PCBEL	子を返す
bottom():PCBEL	末子を返す
elder():PCBEL	兄を返す
little():PCBEL	弟を返す

■使用例：

```
import class.PCBEL;
!w1                                                    // ※ワーニングレベルは１以下であること
class   MyRelation(name:string) includes PCBEL{
        public  method  view()  writeln("Name=",name);
        }
var     p = new MyRelation("パパ");
var     c = new MyRelation("太郎");
var     b = new MyRelation("次郎");
```

```
p.addchild(b);                  // 次郎をパパの末子とする(最初の子なので長男になる)
b.insert(c);                    // 太郎を次郎の兄とする(太郎が長男になる)
p.child().view();               // "Name=太郎"と表示
```

8 相互関係を扱うクラス（Wrapper クラス版）

```
          [  P  ]
            ↑
[  E  ] ← [  X  ] → [  L  ]
            ↓
          [  C  ] .. [  B  ]
```

で示す、多階層にわたる相互関係を扱う。デストラクタによって親が消滅するタイミングでその全ての子が消滅する。

```
import class.Relation

extends Variant
```

コンストラクタ	
Relation(obj:variant)	

インスタンス・メソッド	
insert(obj:Relation):Relation	obj を兄として挿入する
append(obj:Relation):Relation	obj を弟として接続する
inschild(obj:Relation):Relation	obj を長男として接続する
addchild(obj:Relation):Relation	obj を末子として接続する
isparent(obj:Relation):Relation	obj が自身の祖先なら obj を返し、そうでないなら null を返す
ischild(obj:Relation):Relation	obj が自身の子孫なら obj を返し、そうでないなら null を返す
parent():Relation	親を返す
child():Relation	子を返す
bottom():Relation	末子を返す
elder():Relation	兄を返す
little():Relation	弟を返す

■使用例：

```
import class.Relation;
class   MyClass(name:string){
        public  method  view()  writeln("Name=",name);
        }

var     p = new Relation(new MyClass("パパ"));
```

```
var     c = new Relation(new MyClass("太郎"));
var     b = new Relation(new MyClass("次郎"));
p.addchild(b);                          // 次郎をパパの末子とする(最初の子なので長男になる)
b.insert(c);                            // 太郎を次郎の兄とする(太郎が長男になる)
p.child().view();                       // "Name=太郎"と表示
```

9 SMTP による電子メールを扱うクラス

```
import class.SMTP

extends Socket
```

コンストラクタ	
SMTP(server_name:string,port=25)	server_name には SMTP サーバ名を指定する port は既定 port 番号以外（例：587）を使用する場合に指定する

クラス・メソッド	

インスタンス・メソッド	
check(to:string):bool	to:送信先メールアドレス
send(to:string[3], from:string, subject:string, body:string, content_type:string="text/plain" attachimnt:Attachiment[]=null,):string	to[0]:送信先メールアドレス to[1]:Cc メールアドレス to[2]:Bcc メールアドレス from:発進元メールアドレス subject:サブジェクト body:本文 content_type:HTML メールを送信する場合、”text/html”とする attachiment:添付ファイル記述(Attachiment)の配列を指定すると、添付ファイルを送信する Attachiment は、record{field name:string;field body:Buffer}を要素とする配列 送信に成功した場合は空文字列を返し、失敗した場合はその内容を示す文字列を返す ※メールアドレスは、"id@domain" もしくは "ニックネーム <id@domain>" 書式であること。 なお、to[0]、to[1]、to[2]については複数のメールアドレスを指定でき、その場合は";"（セミコロン）で区切る必要がある。 エラーがなければ""を返す。

10POP3 による電子メールを扱うクラス

```
import class.POP3

extends Socket
```

コンストラクタ	
POP3(server_name:string, userid:string, password:string, maxlen:int=8192)	server_name:POP3 サーバ名（ex:"mail.act.ne.jp"） userid:ユーザＩＤ password:パスワード maxlen:最大メッセージバイトサイズ

クラス・メソッド	

インスタンス・メソッド	
getCount():int	受信されているメールの総数を返す
delete(index:int)	index で指定するメールをサーバ内から削除する
read(index:int)	index で指定するメールを読み込む
getHead():string	読み込んだメールのヘッダを返す
getBody():string	読み込んだメールの本文を返す
getKeyItem(key:string):string	ヘッダ中の key で指定する識別データを取り出す
getSubject():string	"SUBJECT:"識別に対応するデータを取り出す
getDate():string	"DATE:"識別に対応するデータを取り出す
getFrom():string	"FROM:"識別に対応するデータを取り出す
getStatus():string	"STATUS:"識別に対応するデータを取り出す
getReturn():string	"RETURN-PATH:"識別に対応するデータを取り出す
close()	メールサーバから切断する

インスタンス・フィールド	
status:string	内部状態を保持する

11HTTP POST を扱うクラス

```
import class.Poster

extends Buffer
```

コンストラクタ	
Poster(multipart:bool=false)	バイナリを post する場合は multipart に true を指定する

インスタンス・メソッド	
addItem(name:string,val:variant)	name で指定するフィールド名と、対応する値として val をキーセットとして追加する
addBinary(name:string, filepath:string, buffer:Buffer)	name で指定するフィールド名と、ファイル名パートに filepath を、バイナリパートに buffer を展開したキーセットを追加する。 ※コンストラクタ引数 multipart を true にしていなければならない
addFile(name:string, filepath:string)	name で指定するフィールド名と、filepath に示すファイルおよびその内容を展開したキーセットを追加する。
post(host:string,path:string) :Buffer	http://<host>/path にキーセットオブジェクトを POST 送信し、レスポンスを返す。

12FIFO を実現するクラス

```
import class.Queue
```

コンストラクタ	
Queue()	Queue オブジェクトを生成する

インスタンス・メソッド	
length():int	キュー内の要素数を返す
enqueue(value:variant)	value をキューに追加する
dequeue():variant	キューの先頭から１要素取り出し、それを返す
push(value:variant)	enqueue()の別名
pop():variant	dequeue()の別名

■使用例：

```
import class.Queue;
var     que:Queue = new Queue();
que.enqueue(100);
que.enqueue("Hello");
while(que.length()){                    // キューを取り出して表示する
        writeln(que.dequeue());
        }
```

13LIFO を実現するクラス

```
import class.Stack
```

コンストラクタ	
Stack()	Stack オブジェクトを生成する

インスタンス・メソッド	
length():int	スタック内の要素数を返す
push(value:variant)	value をスタックに enqueue する
pop():variant	スタックから１要素を dequeue し、その値を返す

■使用例：

```
import class.Stack;
var     stack:Stack = new Stack();
stack.push(100);
stack.push("Hello");
while(stack.length()){                  // スタックを取り出して表示する
        writeln(stack.pop());
        }
```

14Session を管理するクラス

```
import class.Session

extends DBMAN
```

コンストラクタ	
Session()	新たなセッションを開始する場合に使用し、セッション管理オブジェクトを生成する
Session(ID:string=””)	２度目以降のセッション管理オブジェクトを生成する

クラス・メソッド	
dbPath(path:string=””):string	path が””でない場合、セッションＤＢのパスとし、以前のパス値を返す
sweep():int	タイムアウトになったユーザ領域を開放し、有効なユーザ数を返す
remove(id:string)	id で指定するユーザのセッション情報を削除する

インスタンス・メソッド	
isNew():bool	初めてのセッションおよび２度目以降のセッションでタイムアウトの場合に true を返す
getID():string	セッションＩＤ値を返す
getLastAccess():Date	最後にアクセスした日時を返す
setExpire(date:Date)	date に指定する期限を設定する。設定した期限に達したら isNew()が true となる
setTimeout(seconds:int)	タイムアウトに seconds 秒を設定する。seconds 秒内にアクセスがなければタイムアウトとなる
getValue(name:string):variant	name で示すキーセットの値を返す。name キーが登録されていない場合は null を返す
setValue(name:string, value:variant)	name で示すキーセットとして value を登録する。キーセットがなければ新たに生成し、あれば置き換える
resetValue()	すべてのキーセットを破棄する。次回の isNew()は true となる。
resetValue(name:string)	name で示すキーセットを破棄する
close(emptyRemove:bool=true)	セッション管理オブジェクトを閉じる。resetValue()によってすべてのキーセットがクリアされている場合は、remove()を呼び出した場合と同じにセッション終了となる。 ※セッションオブジェクトがデストラクトされるタイミングで自動的に呼び出されるため、明示的に呼び出す必要はない。
remove()	現在のセッションを close()して破棄する。

■使用例：

```
import class.Session;
try{
```

```
        var     session = new Session();                        // セッションを開く(cookie が使える場合のみ)

        if(session.isNew())     session.setValue(“key”,1);      // 最初のセッションならセッション情報を初期化
        var     count = session.getValue(“key”);                // セッションに格納されたセッション情報を取り出す
        writeln(“count=”,count);                                // セッション情報を表示
        count++;  session.setValue(“key”,count);                // セッション情報を更新して保存
        if(count > 3)   session.resetValue();                   // セッションの破棄
        else{
        <FORM><INPUT type=”submit”  value=”次”></FORM>          // 次のセッションへ
        }
        session.close();                                        // セッションを閉じる
}catch(err){
        write(err);     // new Session()を２回実行した（※Session インスタンスはシングルトンでなければならない）
}
```

■ 注意：

document.buffer(false) 状態の場合、new Session() を実行する以前にタグなどを出力してはならない。
new Session()時にヘッダ情報内にセッションＩＤ情報を追加しようとするが、この時点でタグなどが出力されているとヘッダはすでに送出されているため、セッションＩＤ情報がドキュメントの一部として送出されてしまう。

15色を扱うクラス

```
import class.Color
```

コンストラクタ	
Color(r:int,g:int,b:int)	RED、GREEN、BLUE の各値を 0〜255 の値で指定する
Color(rgb:int)	RGB 値を示す整数（Bit0..7:BLUE / Bit8..15:GREEN / Bit16..23:RED）
Color(rgb24:RGB24)	RGB24[1]のインスタンスで指定する
Color(col:Color)	コピーコンストラクタ

インスタンス・メソッド	
getValue():RGB24	RGB24 を返す
toIntValue():int	整数値を返す
toHexString():string	RGB16 進表記文字列を返す(ex:赤=”FF0000”,緑=”00FF00”,青=”0000FF”)
gradation(to:Color,div:int) :Color[]	自身のカラーと指定したカラーの間のグラデーションカラー配列を生成する。 div には分解する要素数(1〜256)を指定する。
operation(ope:string,col:Color) :this	自身のカラーと col に指定したカラーとを演算して自身に反映し、自身への参照を返す。 ope には "~","^","&","|","+","-" の何れかを指定し、以下の演算となる。 “~”: 補色を得る。col は指定の必要が無い “^”: 各色要素を XOR した結果を返す “&”: 各色要素を AND した結果を返す “|”: 各色要素を OR した結果を返す “+”: 各色要素を加算した結果を返す “-“: 各色要素を減算した結果を返す

■使用例：

```
import  class.Color;
var     left = new Color(0xFF,0x80,0x80);
var     right = new Color(new Color.RGB24(0x0080FF));
var     gradArray = left.gradation(right,16);
```

[1] RGB24 クラスは Color クラス内に定義されているサブクラス。 new Color.RGB24(int) でインスタンス生成できる

16統計を扱うクラス

```
import class.Statistics

extends Variant
```

コンストラクタ	
Statistics(num:number[])	
Statistics(obj:Statistics)	コピーコンストラクタ

インスタンス・メソッド	
total():real	合計を返す
average():real	平均を返す
standardDeviation():real	標準偏差を返す　標準偏差 = √((各要素値-平均)の2乗の総和/要素数)
deviation():number[]	偏差値を返す　　個々の要素の偏差値 = (要素値-平均)*10/標準偏差+50

■使用例：

```
import  class.Statistics;
var     st = new Statistics({1,2,3,4,5});
writeln(st.standardDeviation())                 // 標準偏差値を表示する
writeln(st.deviation());                        // 偏差値配列をリスト表示する
```

17統計を扱うクラス（日本語版）

import class.統計

extends Variant

コンストラクタ	
統計(num:number[])	
統計(obj:統計)	コピーコンストラクタ

インスタンス・メソッド	
合計():real	合計を返す
平均():real	平均を返す
標準偏差():real	標準偏差を返す　標準偏差 = √((各要素値-平均)の 2 乗の総和/要素数)
偏差値():number[]	偏差値を返す　　個々の要素の偏差値 = (要素値-平均)*10/標準偏差+50

■使用例：

```
import class.統計;
var     st = new 統計({1,2,3,4,5});
writeln(st.標準偏差())                  // 標準偏差値を表示する
writeln(st.偏差値());                   // 偏差値配列をリスト表示する
```

18期間を扱うクラス

```
import class.Term
```

コンストラクタ	
Term()	
Term(arguments)	arguments は Date コンストラクタ引数と同じ

インスタンス・メソッド	
setStart(arguments):Term	開始日時を設定し、このオブジェクトへの参照値を返す。 arguments は Date コンストラクタ引数と同じ。
setPeriod(arguments):Term	期限日時を設定し、このオブジェクトへの参照値を返す。 arguments は Date コンストラクタ引数と同じ。
getStart():Date	開始日時を返す
getPeriod():Date	期限日時を返す
getSpan():real	期間を実数で返し、整数部で日数、小数部で時刻値の１日を 1.0 とした比率を表す
setSpan(span:real)	span で期間を指定する。span の内容は getSpan()と同じ
addStart(span:real)	開始日時を span だけ加算する。span の内容は getSpan()と同じ
addPeriod(span:real)	期限日時を span だけ加算する。span の内容は getSpan()と同じ
shift(value:real)	期間を維持したまま、開始日時、期限日時に value 値を加算する。value の内容は getSpan()と同じ

19祝日を扱うクラス

```
import extends.Date.Holiday

extends Date
```

コンストラクタ	
Holiday()	現地時刻系で現在の日付時刻を保持する
Holiday(arguments)	arguments は Date コンストラクタ引数と同じ

クラス・メソッド	
list(year:int):HolidayInfo[]	指定した西暦年の祝日リストを返す。 HolidayInfo 型は date:Date フィールドに祝日を、name:string フィールドに祝日名称文字列をもつ。祝日名称は振替休日の場合は"振替休日"となる。 また、toString(format:string):string メソッドを持ち、format に Date クラスインスタンスの toString(format:string)と同じ指定をおこなうと、祝日日付を文字列化した結果に祝日名称を結合した結果を返す。foramt に""を指定した場合は祝日名称のみを返す。 format を指定しない場合は既定値として "$M 月$D 日（$X）" が採用される。

インスタンス・メソッド	
isHoliday():string	祝日なら祝日名を、代替休日なら〝振替休日〟を、それ以外は空文字列を返す

■使用例：

以下は、2001 年の祝日と振替休日をすべて表示する。

```
import  extends.Date.Holiday;
var     list:HolidayInfo[] = Holiday.list(2001);
foreach(item in list){
        writeln(item.toString("$Y/$M/$D ($X)"));
}
```

20和暦を扱うクラス

```
import extends.Date.和暦

extends Date
```

コンストラクタ	
和暦()	現地時刻系で現在の日付時刻を保持する
和暦(arguments)	arguments は Date コンストラクタ引数と同じ

インスタンス・メソッド	
和暦月():string	月名を睦月、如月、、、師走で返す
和暦年():string	該当する明治、大正、昭和、平成の各年で返す

21CSV 形式文字列をセルに分解するクラス

```
import extends.StringBuffer.CSV_Analyzer
```

extends StringBuffer

コンストラクタ	
CSV_Analyzer(text:string)	text に指定する CSV 書式文字列をセル分解して保持する
CSV_Analyzer(buf:Buffer)	buf に指定する CSV 書式文字列をセル分解して保持する
CSV_Analyzer(storage:Storage)	storage に指定する CSV 書式テキストファイルをセル分解して保持する。 storage には new Storage(＜ファイル名文字列＞) のインスタンスを指定する

インスタンス・メソッド	
getValue():Array[]	セル内容を保持する最上位層配列を返す
rowLength():int	セル行数を返す。getValue().length()と同じ
clmLength(row:int):int	指定行のセル数を返す。getValue()[row].length()と同じ
getRow(row:int):string[]	指定行のセル内容を文字列の配列で返す。getValue()[row]と同じ
getCell(row:int,clm:int):string	指定行指定桁のセル内容を返す。getValue()[row][clm]と同じ

■使用例：

"csvfile.csv"という CSV 書式のテキストファイルが用意されているものとする。

```
import  extends.StringBuffer.CSV_Analyzer;
var     csv = new CSV_Analyzer(new Storage("csvfile.csv"));
repeat(csv.rowLength(); row = 0)
        repeat(csv.clmLength(row); clm = 0)
                writeln(csv.noralize(csv.getCell(row,clm)));
```

22テキストファイルを読み出すクラス

```
import extends.Storage.TextFile

extends Storage
```

コンストラクタ	
TextFile(path:string)	path に指定するテキストファイルを開く

インスタンス・メソッド	
loadText(isthrow:bool=false) :string	インスタンスが開いているテキストファイルの内容をすべて返す。 isthrow に true を指定した場合、ファイルがない場合は"TextFile:Not Opened"を例外として発行する。

■使用例：

〝textfile.txt〟というテキストファイルが用意されているものとする。

```
import  extends.Storage.TextFile;
const   path:string = "textfile.txt";
var     txt:string = new TextFile(path).loadText();
```

23文字列を解釈するクラス

```
import class.Parser
```

クラスメソッドを提供する抽象クラス

クラス・メソッド	
DateParse(input:string):Date	input に与えられた日付時刻表記を解釈し、Date インスタンスを返す。 書式にエラーがある場合は null を返す。 入力書式は以下の３書式に対応する。下記にて［＜文字列＞］の記述は ＜文字列＞ を省略可能という意味であり、また記述中のスペース文字の長さに制限はない ■ YYYYY [/] MM [/] DD [hh [:] mm [[:] ss]] 例: "1999/12/31 23:59:59" あるいは "1999 12 31 23 59 59" ■ <monthname> DD YYYY [hh [:] mm [[:] ss]] <monthname>::= <jan[uary] | feb[ruary] | mar[ch] | apr[il] | may | jun[e] | jul[y] | aug[ust] | sep[tember] | oct[ober] | nov[ember] | dec[ember]> 例: "July 21 1999 0:0" あるいは "Jul 21 1999 0:0:0" ※月名の大文字小文字は同一視する ■ <平[成]|昭[和]|大[正]|明[治]> YY <年|/> MM <月|/> DD [日] [hh <時|:> mm <分|:> [ss [秒]]] 例: "平成 11 年 10 月 10 日" あるいは "昭 27 年 2 月 19 日 11 時 23 分"

■ 使用例：

DateParse()の使用例

```
import  class.Parser;
var     data[]:string = {"昭和 27 年 2 月 19 日 23 時 33 分   "," May 19 2006 18: 32   "," 2006/11/30 12: 3  "};
foreach(item in data){
        var     dates:Date = Parser.DateParse(item);
        write(item.trim()," ⇒  ");
        if(dates instanceof Date)       writeln(dates.toString("$Z 年$M 月$D 日 $h 時$m 分$s 秒").replace("1 年","元年"));
        else    writeln(typeof(dates));
        }
```