

Office VBA  
Conference  
2000

General トラック 11:05~12:05

# 『VBA ハイパーテクニック』

**Masato Hirai**

Senior SE

Systems Engineering Group

Microsoft Co., Ltd.

# Session Agenda



はじめに



スーパーテクニック



データ型



VBAの文字列管理



列挙型



Windows API



コレクション



クラスプログラミング



エラー処理とデバッグ



パフォーマンス

# Office VBA Conference 2000

はじめに

VBAプログラミングへようこそ！

# Option Explicit は必須アイテムだ

## Option Explicit とは

- 自動変数を抑止するキーワード
- 宣言しない変数はエラー
- デフォルトで何故か無効

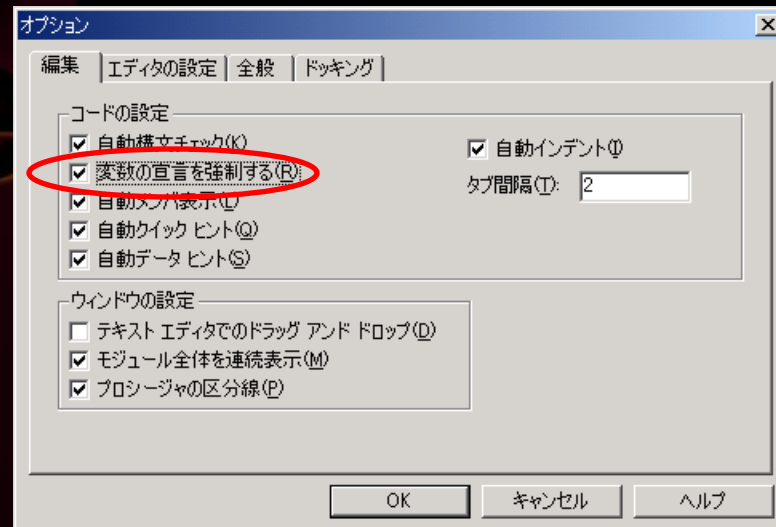
## 必ずつける

- Visual Basic Editor

[ツール]メニューの[オプション]

- 既存のモジュール

モジュールの先頭に Option Explicit と手入力

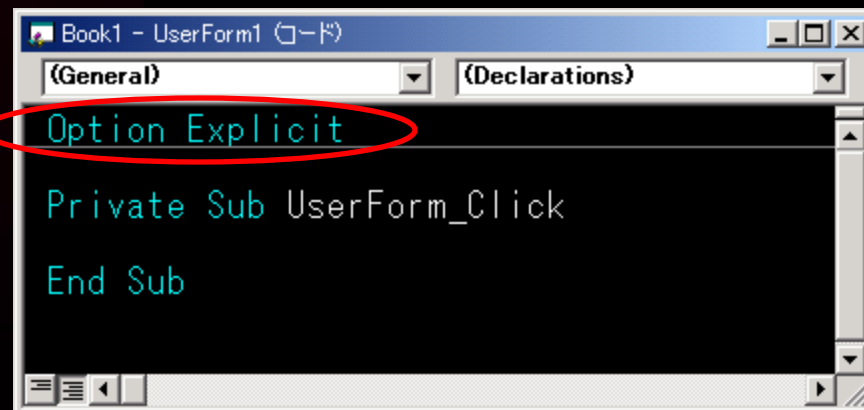


```
tmp = 4  
MsgBox tmp * 9 ' // 36と表示
```



タイプミス

```
tmp = 8  
MsgBox temp * 9 ' // 0と表示
```



VBAプログラミングへようこそ！

# これは便利だー コメント ブロック

🧩 VBAには複数行をコメントアウトする文法がない

- C言語のような `/* ... */` に相当する文法

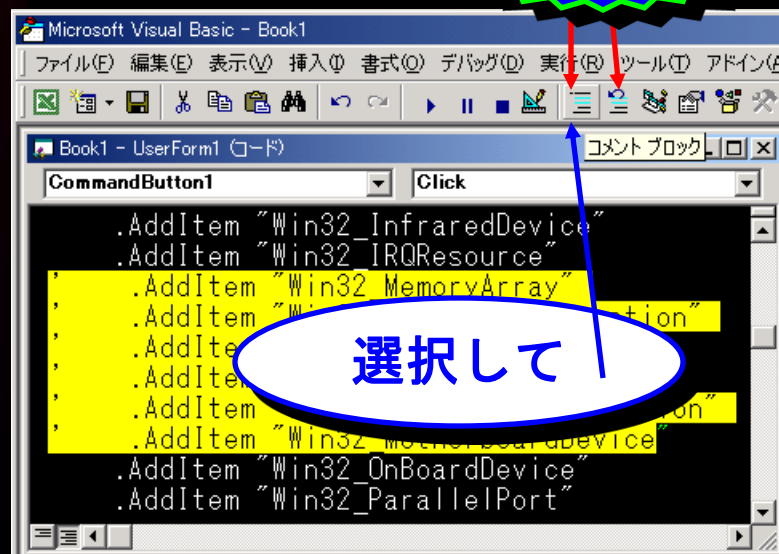
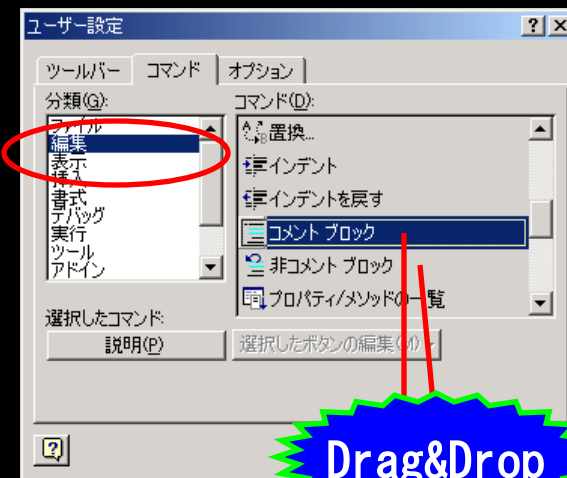
🧩 “コメントブロック”

“非コメントブロック”機能を使おう

- ツールバーのボタン1つで選択した複数行をコメントアウト！
- と言っても行頭に ' を付ける／取るだけ

🧩 方法

- [表示]→[ツールバー]→[ユーザー設定]
- [コマンド]タグの[分類]で“編集”を選択
- “コメント ブロック”と  
“非コメント ブロック”をツールバーへ
- 複数行を選択してボタンを押すだけ



VBAプログラミングへようこそ！

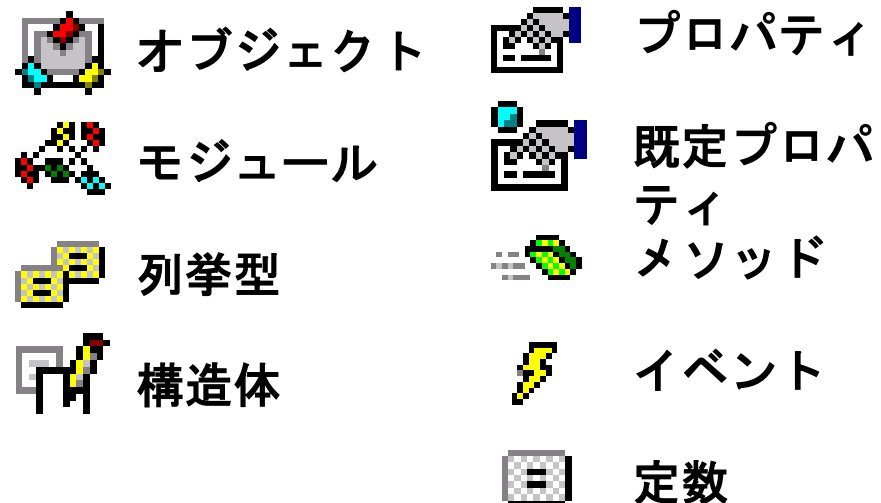
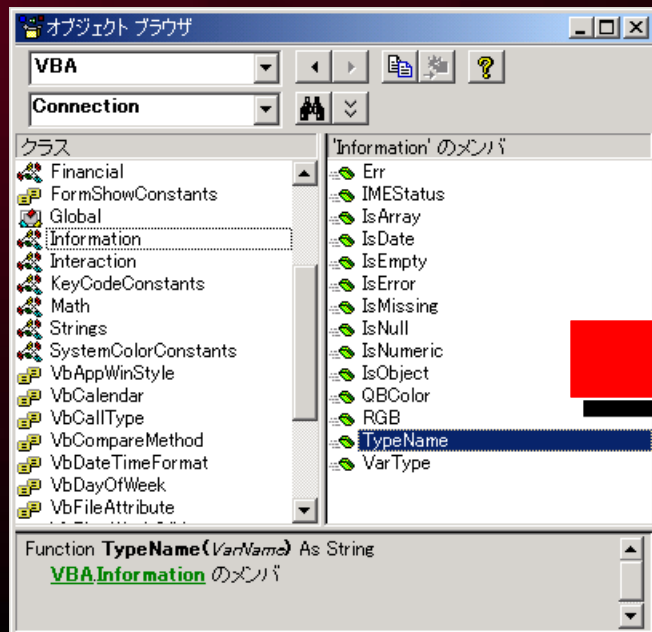
# 何でも見えるぞー オブジェクト ブラウザ

## オブジェクト ブラウザとは

- プロジェクトが参照しているライブラリの詳細を閲覧するツール
- [表示]→[オブジェクト ブラウザ]または[F2]キーで起動

## 何が見える？

- ライブラリ、オブジェクト、メソッド・プロパティ・イベント
- 引数・戻り値のデータ型、構造体・定数・列挙型





VBAプログラミングへようこそ！

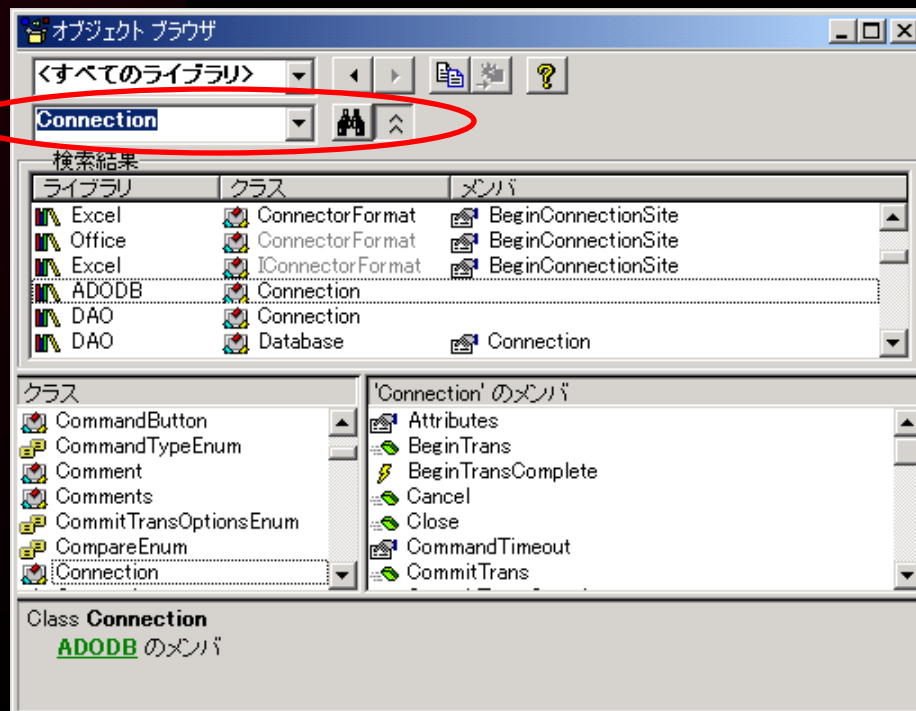
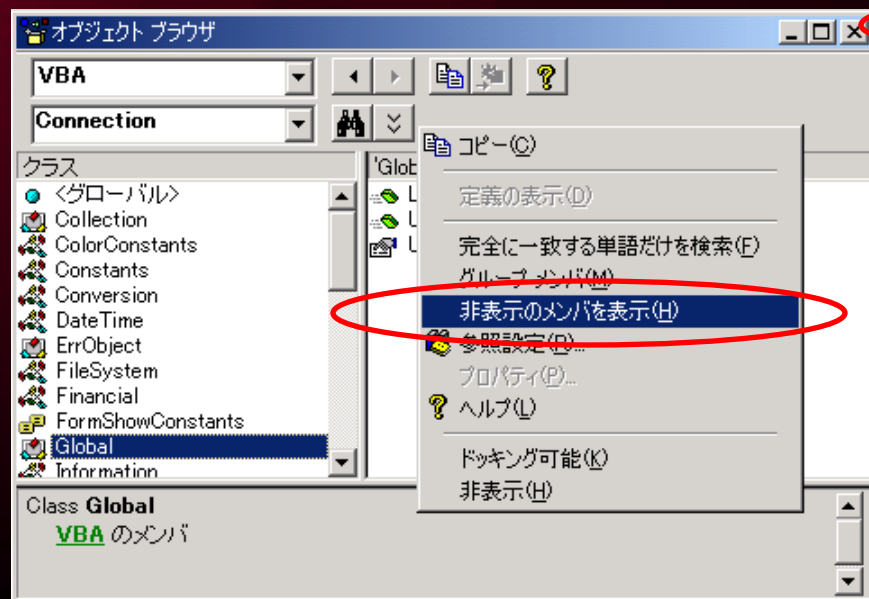
# 隠してもムダー オブジェクト ブラウザ

 隠しオブジェクトやメソッドなども表示可能

- 右クリックメニューの[非表示メンバを表示]をチェック
- エディタ上へも反映される

 オブジェクト ブラウザを活用しよう！

- ヘルプへのジャンプ ([?]ボタンまたは[F1]キー)
- 検索機能つき



VBAプログラミングへようこそ！

# ツールは活用すべし — WinAPI ビューア

 VBAで利用できるほとんどの Windows APIを網羅している

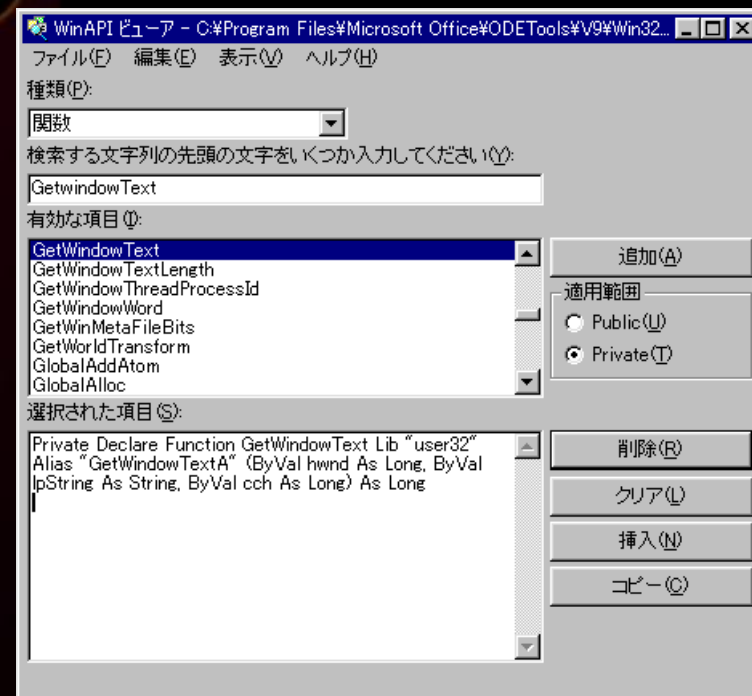
- 宣言 — Declare
- 構造体 — Type
- 定数 — Const

 APIビューアの記述を信用するな

- 必ずしもベストとは限らない
- Platform SDKと合わせて利用
- Wバージョン(Unicode)はない

 オリジナルの作成も可能

- テキストベース(mdbへも変換可能)
- 書式は標準モジュールと同じ
- 既存の WIN32API.TXT を見ればなるほど！
- Windows 2000用、ODBC API用、オリジナルDLL用など...作りましょ  
う！





VBAプログラミングへようこそ！

# コードライブラリアン

## 🧩 ソースコードのデータベース

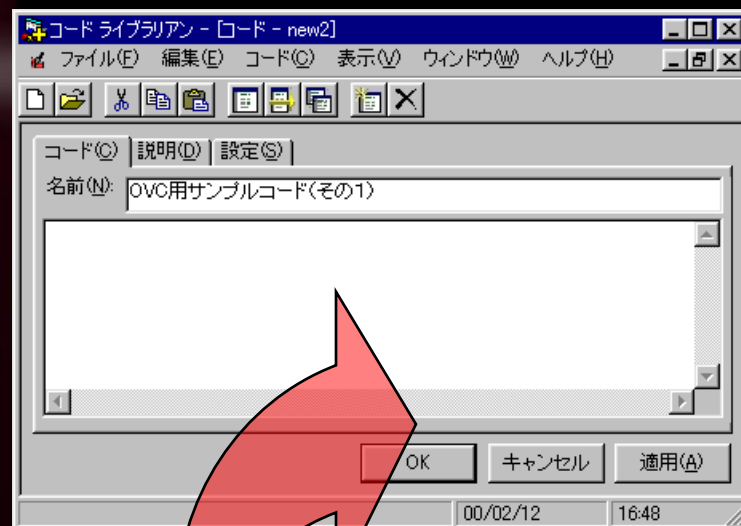
- 再利用性を高める
- Office 2000 Developerに添付

CodeLib.mdb

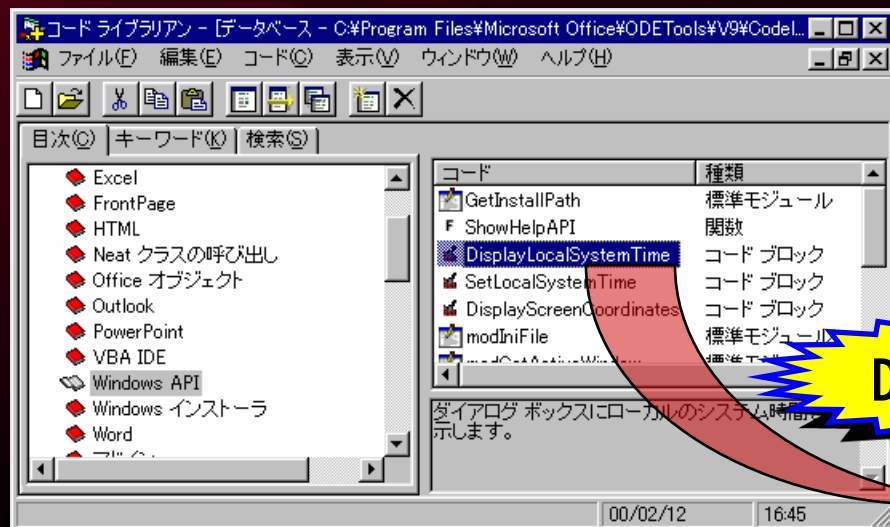
- オリジナルも作ろう！

## 🧩 汎用的なコードは登録しておこう！

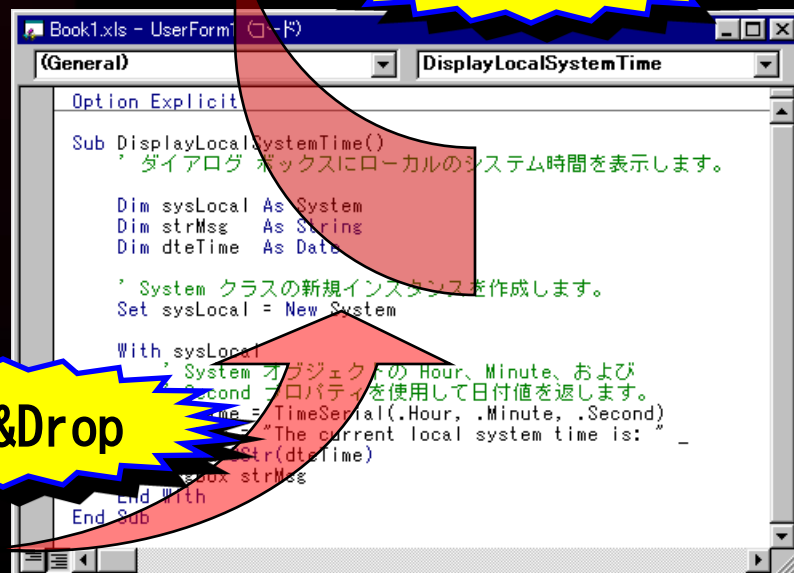
- Office VBAでソースコードの共有は難あり



Cut&Paste



Drag&Drop



VBAプログラミングへようこそ！

# VBAとタイプライブラリのいい関係

## [参照設定] ダイアログボックス

- [ツール]メニューの[参照設定]
- レジストリ[HKEY\_CLASSES\_ROOT¥TypeLib¥]から情報を取得

## 参照設定するとタイプ ライブラリが読み込まれる

VBA (Visual Basic For Applications)

→ VBE6.DLL

Excel (Microsoft Excel 9.0 Object Library)

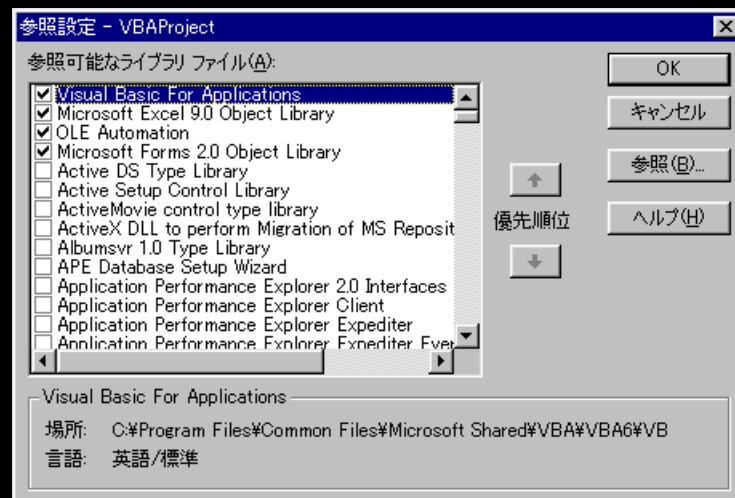
→ EXCEL9.OLB

stdole (OLE Automation)


→ STDOLE2.TLB

## [参照] ボタンで指定すると レジストリに登録される

- DLL, TLB, OLE, EXE を指定
- レジストリに登録されていないものは自動的に登録される
- レジストリから削除するには  
REGSVR32 /u xxxxxx.dll



# タイプライブラリは外界への案内窓口

 データ型・モジュール・インターフェイスの標準的な定義が記述されたファイル（リソース）

- オブジェクト・インターフェイス・モジュール
- データ型・列挙型・構造体・定数
- メソッド・プロパティのパラメータや戻り値についての情報
- ヘルプファイル（コンテキスト）や説明など

 タイプライブラリの形態

- 拡張子は通常 \*.TLB
- オブジェクト ライブラリ (\*.OLB)
  - 1 つまたは複数のタイプライブラリ リソースを持つ DLL

単独のバイナリファイル (TLB)

オブジェクト ライブラリ (OLB)

公開

DLLのリソース (DLL)

EXEのリソース (EXE)

# Office VBA Conference 2000

## データ型


データ型を制覇する

データ型を制覇する

# ちょっと待った！変数宣言その前に

 変数は必ず型を宣言すること！

- 型を宣言しない変数はバリエーション型
- *Def Type* により既定のデータ型を変更可能

 オブジェクト変数は  
“ライブラリ名.オブジェクト名”で！

- 同じオブジェクト名が存在するケース
- [参照設定]で指定した順序が優先
- 定数・構造体などにも適用

```
Dim cnDAO As DAO.Connection
Dim cnADO As ADODB.Connection
Dim cn As Connection
```

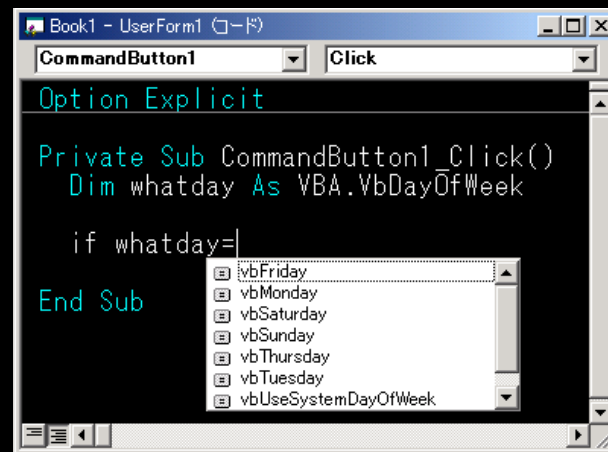
 列挙型 (Enum) も使える！

- 実際のデータ型は長整数 (Long) 型
- インテリジェンス機能の恩恵を受ける

```
Dim humpty As Long
Dim dumpty
humpty = 123456789
dumpty = 123456789
MsgBox LenB(humpty)
MsgBox LenB(dumpty)
```

```
Option Explicit
DefInt A-Z
```

```
Dim whatday As VBA.VbDayOfWeek
```





データ型を制覇する

# 定数には型宣言文字をつけよ！

 コード中の定数や数値リテラルは“型宣言文字”をつける

- データ型を明示する目的
- オーバーフローや演算誤差の防止
- %, &, !, #, @, \$, など

 型宣言をしていない定数や数値リテラルの解釈

- 整数 : -32767 ~ 32767 → Integer型
- : -2147483647 ~ 2147483647 → Long型
- : その他 → Double型
- 実数 : すべてDouble型

```
Dim v As Variant
v = 32767
MsgBox TypeName(v) ' // Integer
v = 32767&
MsgBox TypeName(v) ' // 型宣言文字
v = 32768
MsgBox TypeName(v) ' // Long
```

```
Dim x As Long
x = 32767 + 1 ' // × オーバフロー
x = 32767& + 1 ' // ○
x = 32767 + 1& ' // ○
x = 32767& + 1& ' // ◎
```

# Office VBA Conference 2000

列举型

知られざる機能 ～Enum型～

知られざる機能 ~Enum型~

# Enumを理解する

## 🧩 列挙型

- Const をグループ化したようなもの
- データ型はLong値(負の値も可)

## 🧩 オブジェクト ブラウザで確認

- VBA にも非常に多く定義されている

## 🧩 インテリジェンス機能の恩恵

- 自動メンバ表示

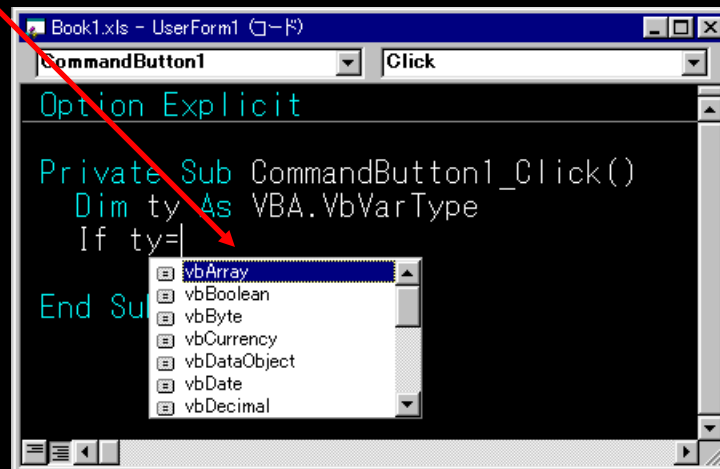
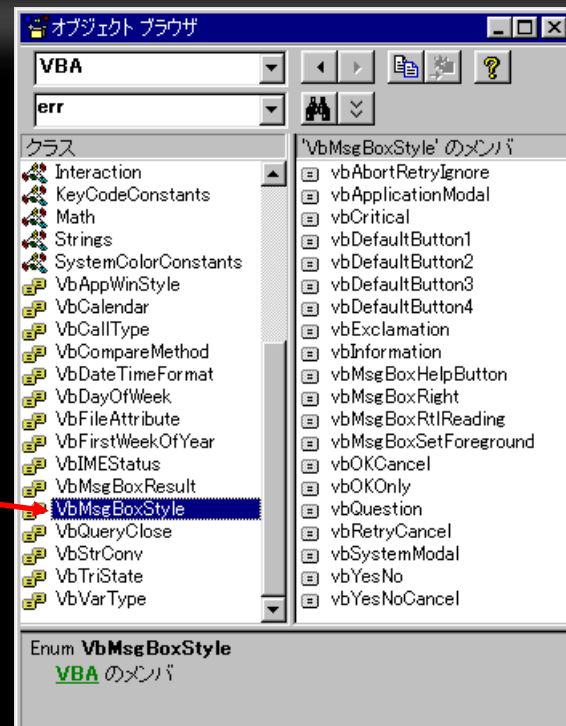
## 🧩 Enumステートメントの構文

[Public|Private] Enum EnumName

membername [= value]

membername [= value]

End Enum



# Enum宣言 — いろいろあるよ

①

```
Enum apPayTypes  
  apCash  
  apCheck  
  apAmEx  
  apMaster  
  apVisa  
  apNone  
End Enum
```

②

```
Enum apPayTypes  
  apCash = 1  
  apCheck  
  apAmEx = 100  
  apMaster  
  apVisa  
  apNone = 999  
End Enum
```

③

```
Enum apPayTypes  
  apCash  
  apCheck = 0  
  apAmEx = 10  
  apMaster  
  apVisa = 10  
  apNone  
End Enum
```

④

```
Public Enum apPayTypes  
  Cash = 1  
  Check = 2  
  [American Express] = 3  
  [Master Card] = 4  
  Visa = 5  
  None = 999  
End Enum
```

⑤

```
Public Enum apPayTypes  
  Cash = 1  
  Check = 2  
  [American Express] = 3  
  [Master Card] = 4  
  Visa = 5  
  [_None] = 999  
End Enum
```

# Enumをどこで使う？

## 変数の定義

```
Dim lngGender As apGender  
lngGender = apFemale
```

## プロパティの値

## プロシージャの引数・戻り値

```
Enum  
Enum apGender  
    apMale = 1  
    apFemale = 2  
    apChild = 9  
    [_apVIP] = -1  
End Enum
```

```
Function AdmissionFee (ByVal Gender As apGender,  
                      ByVal Fee As Currency) As Currency  
    Select Case Gender  
        Case apMale : AdmissionFee = Fee * 1@ ' // 男性:割引なし  
        Case apFemale : AdmissionFee = Fee * 0.8@ ' // 女性:80% OFF  
        Case apChild : AdmissionFee = Fee * 0.5@ ' // 子供:50% OFF  
        Case [_apVIP] : AdmissionFee = 0 ' // VIP :タダ  
        Case Else  
            Err.Raise vbObjectError + 512, , "無効な引数"  
        End Select  
    End Function
```



# ここまでやったらEnumオタク

```
Declare Function SHFormatDrive Lib "shell32" ( _  
    ByVal hWnd As Long, _  
    ByVal DriveID As Long, _  
    ByVal FormatID As shfmtID, _  
    ByVal Options As shfmtOpt) As shfmtErr
```

```
Enum shfmtID  
    SHFMT_ID_DEFAULT = &H0 ' // デフォルト  
    SHFMT_ID_144 = &H3 ' // 1.44MB  
    SHFMT_ID_720 = &H5 ' // 720KB  
End Enum
```

```
Enum shfmtOpt  
    SHFMT_OPT_QUICKFORMAT = &H0 ' // クイック フォーマット  
    SHFMT_OPT_FULL = &H1 ' // 通常のフォーマット  
    SHFMT_OPT_SYSONLY = &H2 ' // 起動専用  
End Enum
```

```
Enum shfmtErr  
    SHFMT_ERROR = &HFFFFFFFF ' // エラー  
    SHFMT_CANCEL = &HFFFFFFFE ' // キャンセル ボタンを押した  
    SHFMT_NOFORMAT = &HFFFFFFFD ' // OSが使ってたるとき  
End Enum
```

# Office VBA Conference 2000

コレクション

何でも集めよう ～コレクション～

何でも集めよう ～コレクション～

# コレクションを知ってるかい？

🧩 関連する項目の集合をグループ化したもの

➤ フォーム上のコントロールのコレクション

UserForm1.Controls

➤ Excelワークブック内のシートコレクション

Workbooks("Book1").Sheets

🧩 配列の代替手段としての利用

➤ 構造体を除くすべてのデータ型を格納

➤ ReDim 不要でメンテナンスがよい

➤ 数値インデックスと文字列キーによる検索

MyCollection(10) = "ABC"

MyCollection("key1") = "ABC"

※コレクションによっては数値インデックスのみ (ex. Forms)

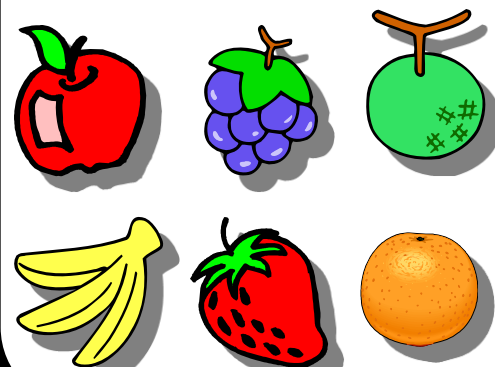
🧩 コレクションは複数形の命名規則 ～S

➤ TableDefオブジェクト ↔ TableDefsコレクション

➤ Documentオブジェクト ↔ Documentsコレクション

Fruits

フルーツ コレクション



何でも集めよう ～コレクション～

# まずはコレクション オブジェクトから

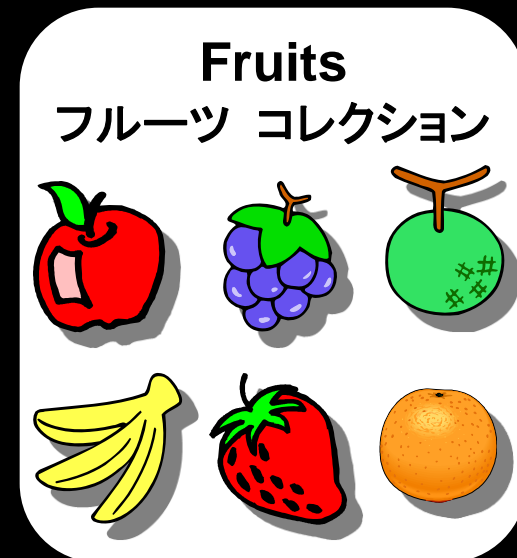
## VBAが提供するオブジェクト

- VBA. Collection
- 追加、削除、検索の基本機能を提供

## バリエーション型で格納される

- 1 要素あたり常に16バイトを使用
- インデックスは 1 から (one-based)

## プロパティとメソッド



Add(Item, [Key], [Before], [After])	項目の追加
Remove(Index)	項目の削除
Count()	コレクションの項目数
Item(Index)	項目を返す (規定)
_NewEnum()	列挙子を返す (For Each)

何でも集めよう ～コレクション～

# コレクションを作る

## コレクションの作成と項目の追加

Dim Fruits As New Collection  
With Fruits

```
.Add "リンゴ", "Apple"  
.Add "ブドウ", "Grape"  
.Add "メロン", "Melon"  
.Add "バナナ", "Banana"  
.Add "イチゴ", "Strawberry"  
.Add "ミカン", "Orange"
```

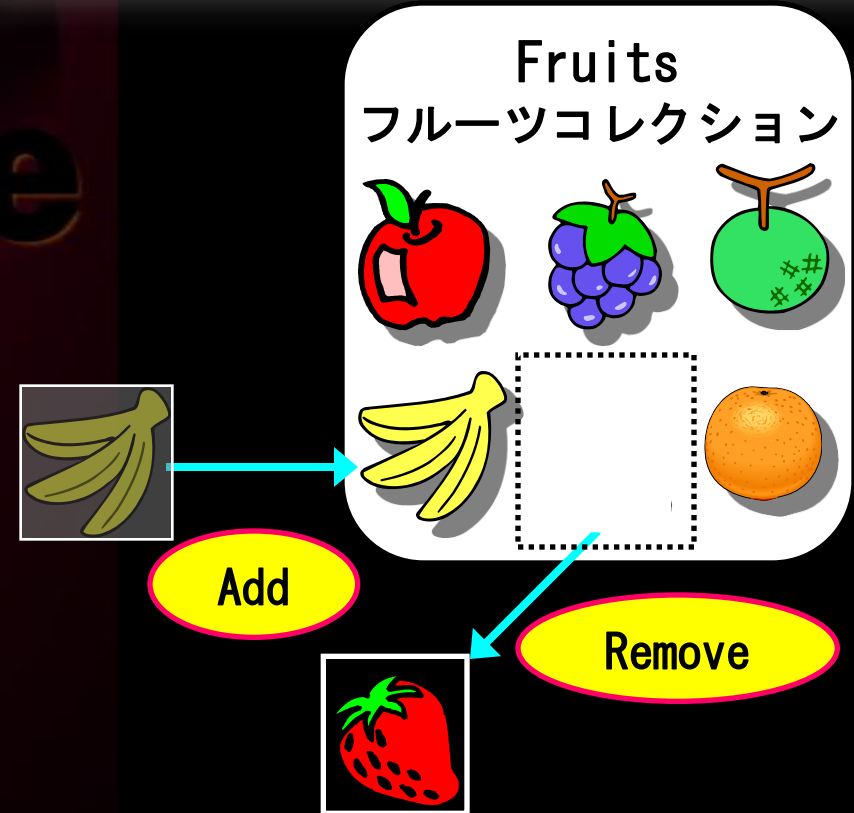
End With

### 追加する項目

- ・ 型はなんでもOK

### 文字列キー

- ・ 索引となる
- ・ 省略可能



## コレクションから項目の削除

```
'// 文字列キー  
Fruits.Remove "Strawberry"  
'// 数値インデックス  
Fruits.Remove 5
```



何でも集めよう ～コレクション～

# コレクションを使う

## Fruits コレクション

```
Dim Fruits As New Collection  
With Fruits
```

```
.Add "リンゴ", "Apple"  
.Add "ブドウ", "Grape"  
.Add "メロン", "Melon"  
.Add "バナナ", "Banana"  
.Add "イチゴ", "Strawberry"  
.Add "ミカン", "Orange"
```

```
End With
```

## Fruits フルーツコレクション



```
s = Fruits.Item(1)  
s = Fruits.Item("Apple")  
s = Fruits(1)  
s = Fruits!Apple  
s = Fruits![Apple]
```

## ① For Each ～ Next

```
Dim fruit As Variant
```

```
For Each fruit In Fruits  
    ListBox1.AddItem fruit  
Next
```

## ② For ～ Next

```
Dim k As Long
```

```
For k = 1 To Fruits.Count  
    ListBox1.AddItem Fruits(k)  
Next
```

# Office VBA Conference 2000

## エラー処理とデバッグ

鳴くに鳴けないホトギス ～エラー処理&デバッグ～

鳴くに鳴けないホトギス ～エラー処理&デバッグ～

# 飛ばぬ先のエラー処理

🧩 エラーのないコードを作成するのが大前提！

➤ 安定したコードを記述(当たり前)

🧩 エラーを制御しよう

On Error ステートメント

On Error Goto Error\_Label

On Error Resume Next

On Error Goto 0

Err オブジェクト

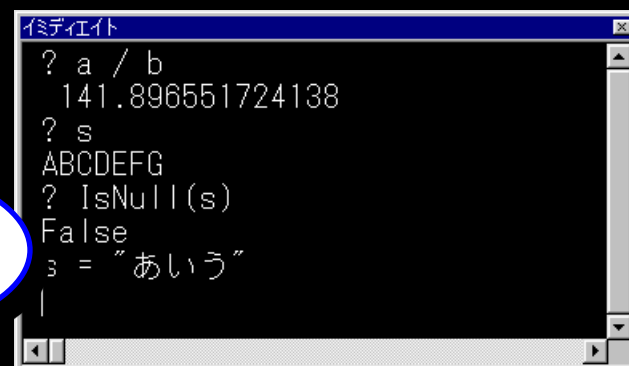
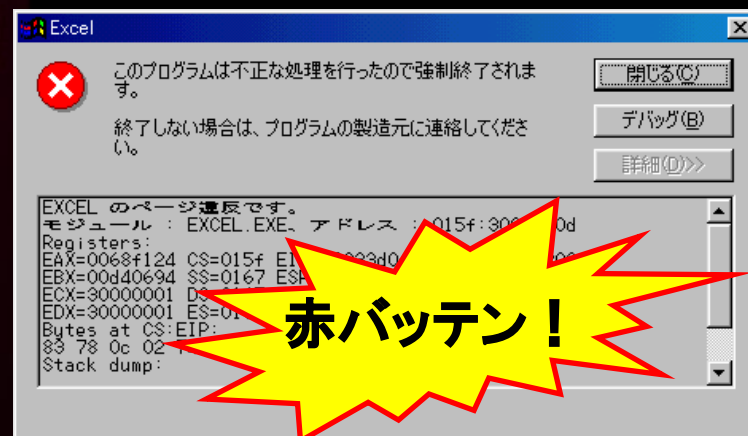
🧩 Debug オブジェクト

Print メソッド

Assert メソッド

🧩 条件付きコンパイル

➤ #If ... Then ... #Else ... #End If ディレクティブ



イミディエイト  
ウィンドウ

# ちょっと止まって下さいーアサーション

## Debug.Assert (boolean expression)

- 「式がTrueの時に中断」のウォッチ式に似ている
- 恒久的なブレークポイントとも言える  
ブレークポイントは保存されない

## アサーションでブレークする2つの条件

- Assert メソッドの条件式が False になるとき
- デザイン環境 (= 開発環境) のとき

## Stop ステートメントと Assert メソッド

- Debug.Assert False  $\doteq$  Stop

```
Private Sub CommandButton1_Click()
```

```
    Debug.Assert False
```

```
    Stop
```

```
End Sub
```

デザイン環境のみ停止

常に停止

鳴くに鳴けないホトギス ～エラー処理&デバッグ～

# どこへいく? On Error Goto xxxxx

```
a = 10
' // エラートラップ開始
On Error GoTo foo_ErrorHandler
JUMP_A:
a = a + b
a = a / b
a = a * b
foo = a
Exit Function

' // エラー処理ルーチン
foo_ErrorHandler:
Debug.Print Err.Number, Err.Description
Err.Clear
b = 2
Resume ' ①
Resume Next ' ②
GoTo JUMP_A ' ③
Exit Function ' ④
```

エラー!

① = 10

② = 20

③ = 12

④ = 0



# Office VBA Conference 2000

スーパーテクニック

これぞ！スーパーテクニック

これぞ！スーパーテクニック

# 核兵器とも言える関数 — VarPtr, StrPtr

 VarPtr – 変数の先頭アドレスを返す

➤ pt = VarPtr(x)

 StrPtr – 文字列の先頭アドレス返す

➤ lp = StrPtr(s)

 ドキュメント化されていない関数

➤ オブジェクト ブラウザで確認(“非表示メンバを表示”をオン)

➤ VBA. [\_HiddenModule]. StrPtr()

API (DLL) や  
メモリ操作で  
強力な武器！

```
Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" ( _  
    ByVal Destination As Long, ByVal Source As Long, ByVal Length As Long)  
' // -----  
Dim a As Integer, b As Integer  
a = 2000  
CopyMemory VarPtr(b), VarPtr(a), 4&  
Debug.Print b
```

これぞ！スーパーテクニック

# LSet って凄いねえ

## 🧩 構造体どうしのコピー

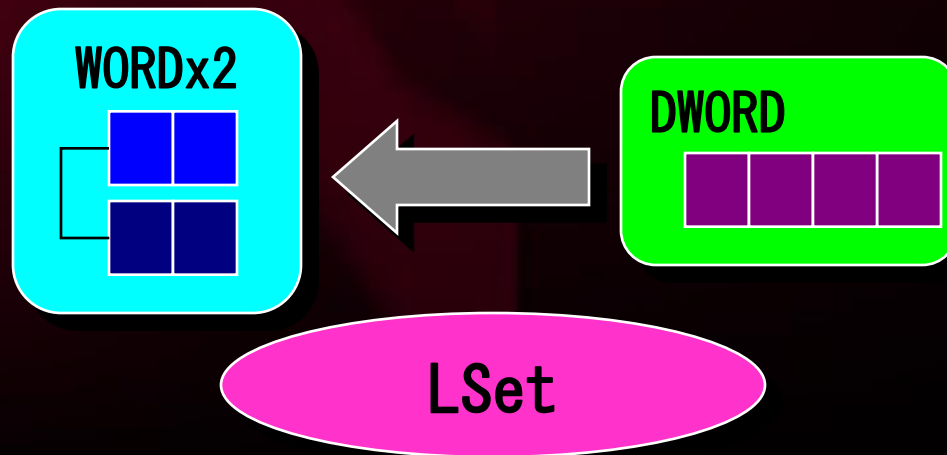
- 左辺・右辺とも構造体

LSet rcMySize = rcYourSize

- 構造体の長さや型は関係ない

## 🧩 利用

- 上位・下位バイトを分ける
- 構造体どうしのコピー
- Windows API にも応用



```
Type WORDx2
```

```
    Lo      As Integer
```

```
    Hi      As Integer
```

```
End Type
```

```
Type DWORD
```

```
    data    As Long
```

```
End Type
```

```
Type BYTEx4
```

```
    data(0 To 3) As Byte
```

```
End Type
```

```
'// -----
```

```
Dim param      As DWORD
```

```
Dim w2         As WORDx2
```

```
Dim b4         As BYTEx4
```

```
param.data = &HAABBCCDD
```

```
LSet w2 = param
```

```
Debug.Print Hex$(w2.Hi)
```

```
Debug.Print Hex$(w2.Lo)
```

```
LSet b4 = param
```

```
Debug.Print Hex$(b4.data(0))
```

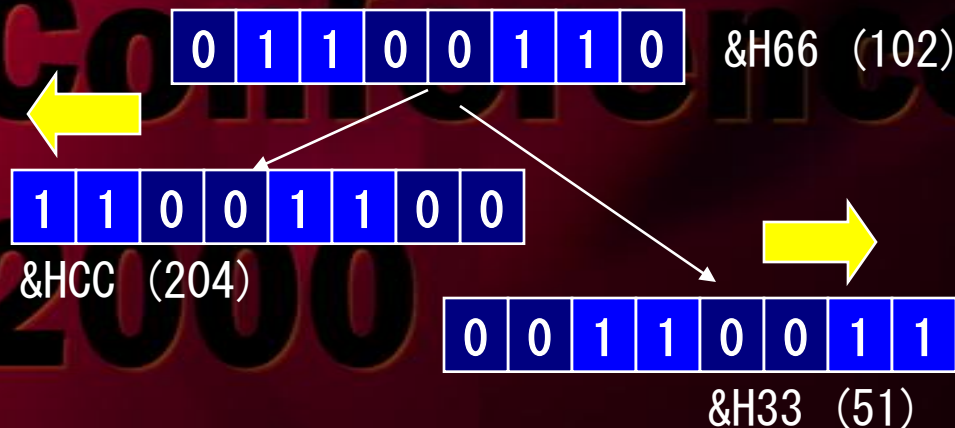
```
Debug.Print Hex$(b4.data(1))
```

```
Debug.Print Hex$(b4.data(2))
```

```
Debug.Print Hex$(b4.data(3))
```

# 右へ左へーシフト演算テクニック

## 🧩 VBAにはシフト演算子がない



## 🧩 2で割れば右シフト

- xをnビット分だけ右シフトする  
 $x = x \div (2^n)$

## 🧩 2を掛ければ左シフト

- xをnビット分だけ左シフトする  
 $x = x * (2^n)$

## 🧩 演算時は大きいデータ型に昇格

- オーバーフローを回避する目的
- Byte→Integer→Long→Currency

```
Private Type BYTEx2
    data      As Byte
    dummy     As Byte
End Type
```

```
Private Type WORDx1
    data      As Integer
End Type
```

```
Function ByteShift(_
    ByVal tg As Byte, _
    ByVal op As String, _
    ByVal st As Long) As Byte
    Dim wide As WORDx1
    Dim mini As BYTEx2
    mini.data = tg
    LSet wide = mini
    Select Case op
        Case ">>" ' // 右シフト
            wide.data = wide.data \ (2 ^ st)
        Case "<<" ' // 左シフト
            wide.data = wide.data * (2 ^ st)
        Case Else
    End Select
    LSet mini = wide
    ByteShift = mini.data
End Function
```

これぞ！スーパーテクニック

# Unicodeを自由自在にあやつる

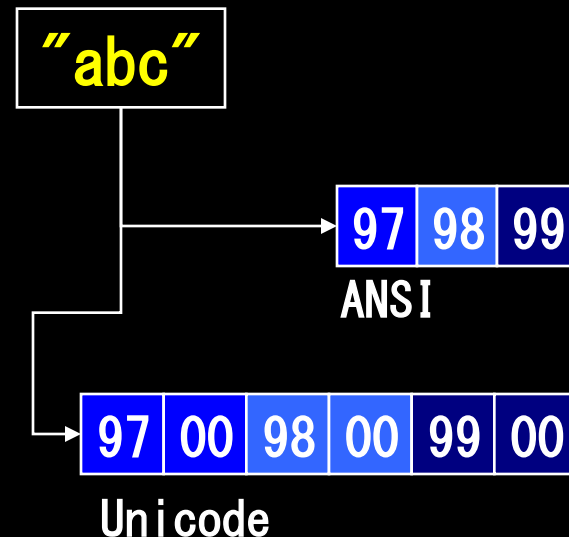
 VBAは文字列をUnicodeで管理

- Unicode：すべての文字を2バイトで表現
  - ANSI：1バイトで表現（DBCSは2バイト）
- ※APIやファイル出力時にANSIに強制変換

 UnicodeはByte配列で扱う！

- String型 ↔ Byte配列の変換は簡単

 Unicode-ANSI変換は StrConv関数で！



```
Dim s As String
Dim b() As Byte
Dim n As Variant
s = "abc"
b() = s ' // 文字列 → バイト配列
For Each n In b()
    Debug.Print n; " ";
Next
s = b() ' // バイト配列 → 文字列
```

```
Dim s As String, b() As Byte, n As Variant
s = "abc"
b() = StrConv(s, vbFromUnicode) ' // ANSI →
For Each n In b()
    Debug.Print n; " ";
Next
s = b()
s = StrConv(b(), vbUnicode) ' // Unicode →
Debug.Print s
```

# Office VBA Conference 2000

## VBAの文字列管理

VBAの内部メカニズムにメスを入れる

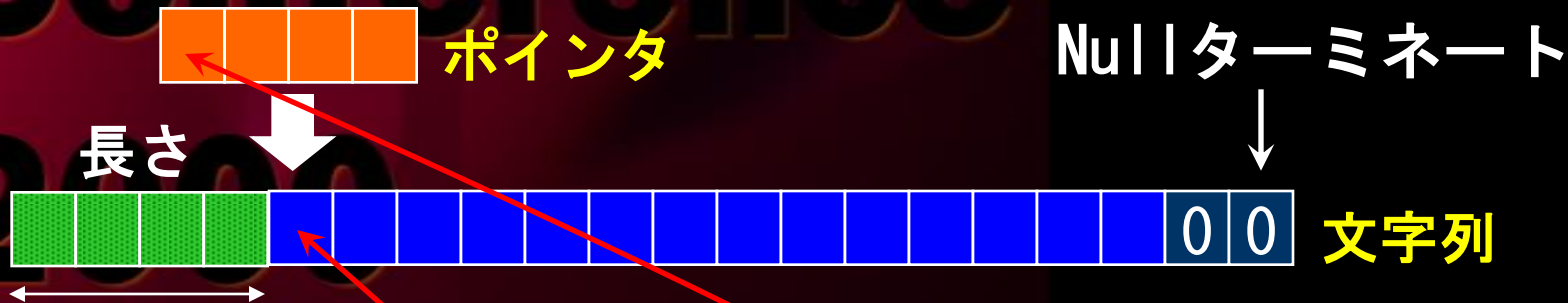


VBAの内部メカニズムにメスを入れる

# こうなっています — VBAの文字列構造

🧩 BSTRフォーマット

(VBA, VB4以降, COM, 32Bit)



🧩 VarPtrとStrPtrが指し示す先

```
Dim s As String  
s = "ABCDEFGH"
```

```
Print VarPtr(s)
```

// ポインタのアドレス

```
Print StrPtr(s)
```

' // 文字列の先頭アドレス

# とっても簡単 — VBAの文字列管理

## 🧩 文字列領域は常に変動する

- APIの呼び出し中は動かない
- ポインタを預けておくAPIは危険
- このような場合はバッファ領域もAPIで確保

## 🧩 文字列は操作する度に別の領域を確保

- 同じ長さの代入でも格納領域が変わる
- 非常にオーバーヘッドが大きい！

```
Dim s As String
s = "ABC"           ' // ①
Debug.Print Hex$(VarPtr(s)), Hex$(StrPtr(s))

s = "XYZ"           ' // ②→③
Debug.Print Hex$(VarPtr(s)), Hex$(StrPtr(s))

s = s & "PQR"       ' // ④→⑤→⑥
Debug.Print Hex$(VarPtr(s)), Hex$(StrPtr(s))
```

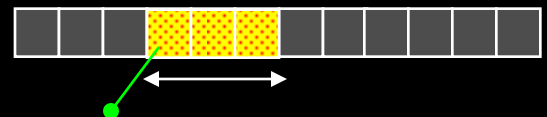
### ① 領域を確保



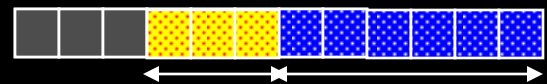
### ② 別領域を確保



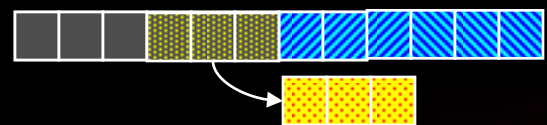
### ③ ポインタを変更



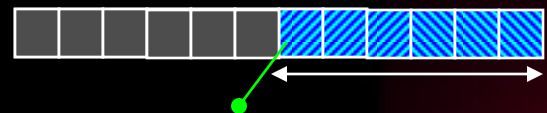
### ④ 別領域を確保



### ⑤ コピー&追加



### ⑥ ポインタを変更



VBAの内部メカニズムにメスを入れる

# 無駄が多い? バリエーション型の内部構造

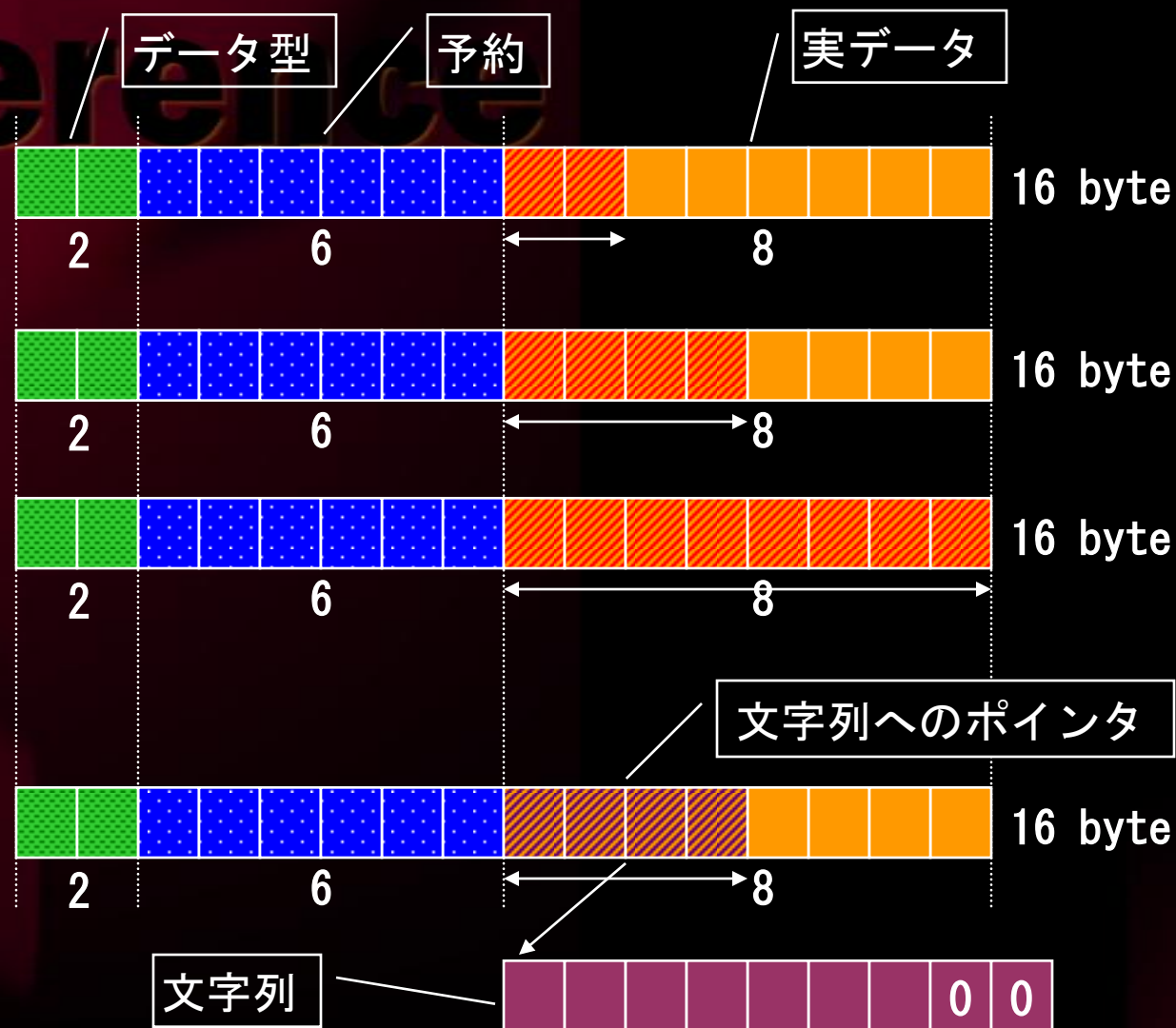
 vbInteger

 vbLong  
vbSingle

 vbCurrency

 vbDouble

 vbString



# Office VBA Conference 2000

## Windows API

不可能を可能にする Windows API

# こんなDeclare宣言ができるんだ！

## Optional キーワードとデフォルト値を使った宣言

- Optional は最後の引数から設定し、デフォルト値も定義

```
Declare Function PlaySoundEx Lib "winmm.dll" Alias "PlaySoundA" ( _  
    ByVal lpzName As String, _  
    Optional ByVal hModule As Long = 0&, _  
    Optional ByVal dwFlags As PLAYSOUND_FLAGS = &H20000) As Long
```

## Alias キーワードを使った宣言

- 関数に対しての別名を指定するときに必要

```
Declare Function PlaySound Lib "winmm.dll" Alias "PlaySoundA" (...
```

- VB で有効な関数名でない場合

```
Declare Function lopen Lib "kernel32" Alias "_lopen" (...
```

- 序数を使って API (DLL) を呼び出す場合

```
Declare Function SetWindowText Lib "user32" Alias "#532" (...
```

OSによって序数が違うこともある。関数名がない場合や軽量化  
DUMPBIN.EXEで取得 → DUMPBIN /EXPORTS user32.dll

# API 呼び出しの知識

## ByVal と ByRef の違いと理解する

- ByVal (値渡し) は値そのものをスタックに積む  
文字列はByValで (VBA側で特殊な処理)
- ByRef (参照渡し) は変数のアドレスをスタックに積む  
API側で LPxxxxと記述されていたら ByRefだ



## 文字列の Nullポインタ

- API (DLL) 側から見れば 0&を要求
- vbNullString という定数を使用

vbNullString = 初期化されていない可変長文字列変数  
As Any で型チェックを放棄して ByVal 0& でも可能  
初期化されていない可変長文字列変数でも可能



```
Declare Function FindWindow Lib "user32" Alias "FindWindowA" ( _  
    ByVal lpClassName As String, ByVal lpWindowName As String) As Long  
' // -----  
Dim s As String  
Debug.Print FindWindow(vbNullString, "電卓"), FindWindow(s, "電卓")
```



# Windows API 呼び出しのメカニズム

## 🧩 呼び出し前の処理



## 🧩 呼び出し後の処理



元の文字列の長さは変わらない (Null処理を余儀なくされる)  
APIが文字列を戻さなくても処理は行われる (間抜け)

# メモリを守れ — APIから文字列の受取り

## 受け取りバッファは十分な長さを確保

- 大きすぎるとオーバーヘッドも大きくなる
- API (DLL) に渡す前に String\$関数で vbNullChar [= Chr\$(0)] を充填
- Nullを考慮してサイズは MAX + 1 にする
- バッファサイズを指示するときは確保したサイズ

```
Dim buf As String  
buf = String$(261, vbNullChar) ' // 領域確保 & Null 充填
```

## 呼び出し後の文字列の整形

- Null文字 (Chr\$(0) = vbNullChar) の切り捨て
- API (DLL) から返される“コピーされたバイト数”は注意
- InStrで Null位置を求め、Left\$関数での切り出し方法がベスト

```
Dim s As String  
s = Left$(buf, InStr(buf, vbNullChar) - 1) ' // 文字列の切り出し
```

# Unicode API に挑戦！

 パターン1. Byte配列へ文字列を格納して呼び出す

- Declareを変更: ByVal s As String → ByRef b As Any
- 自分で Nullターミネートしてバイト配列の先頭を渡す

 パターン2. StrPtrで文字列のポインタを直接送る

- Declareを変更: ByVal s As Long


```
' // パターン1
Declare Function GetShortPathNameW _
    Lib "kernel32" ( _
    IpszLongPath As Any, _
    IpszShortPath As Any, _
    ByVal cchBuffer As Long) As Long
' // -----
Dim lg() As Byte
Dim sh() As Byte
Dim s As String
lg() = "マイクロソフト.txt" & vbNullChar
sh() = String$(261, vbNullChar)
GetShortPathNameW lg(0), sh(0), 261
s = sh()
```

```
// パターン2
Declare Function GetShortPathNameW _
    Lib "kernel32" ( _
    ByVal IpszLongPath As Long, _
    ByVal IpszShortPath As Long, _
    ByVal cchBuffer As Long) As Long
' // -----
Dim lg As String
Dim sh As String
lg = "マイクロソフト.txt"
sh = String$(261, vbNullChar)
GetShortPathNameW StrPtr(lg), _
    StrPtr(sh), _
    261
```

# 64ビット整数 LONG\_INTEGER も怖くない

 VBAには64ビット整数型が存在しない

- 2つのLongメンバを持つ構造体を使う？

 LONG\_INTEGERには通貨型 (Currency) 型が便利！

- データ長が同じ8バイト(64ビット)で内部メモリ形式も同じだ
- ただし、表示形式が固定小数点(小数以下4桁)
- 10000@を掛けたり、割ったりの処理が必要！

```
Declare Function GetDiskFreeSpaceEx Lib "kernel32" Alias "GetDiskFreeSpaceExA" ( _  
    ByVal lpDirectoryName As String, _  
    lpFreeBytesAvailableToCaller As Currency, _  
    lpTotalNumberOfBytes As Currency, _  
    lpTotalNumberOfFreeBytes As Currency) As Long  
' // -----  
Dim AvailableBytes As Currency  
Dim TotalBytes As Currency  
Dim FreeBytes As Currency  
GetDiskFreeSpaceEx "C:", AvailableBytes, TotalBytes, FreeBytes  
Debug.Print AvailableBytes * 10000@  
Debug.Print TotalBytes * 10000@  
Debug.Print FreeBytes * 10000@
```

# やりたい放題 — コールバック関数

 コールバック APIには AddressOf 演算子が活躍する

➤ プロシージャのアドレスが必要なAPI

コールバック関数は標準モジュールに定義する

引数などは Platform SDK で確認する

➤ EnumXXXX系のAPI, ウィンドウプロシージャなど  
サブクラス化に利用

```
' // コールバック関数 (標準モジュールに定義)
```

```
Function WindowProc(ByVal hw As Long, ByVal uMsg As Long, _  
                    ByVal wParam As Long, ByVal lParam As Long) As Long
```

```
End Function
```

```
' // AddressOf演算子を使う
```

```
Private Sub CommandButton1_Click()
```

```
    hWinProc = SetWindowLong(GetActiveWindow(), GWL_WNDPROC, _  
                             AddressOf WindowProc)
```

```
End Sub
```

# Office VBA Conference 2000

## クラスプログラミング

A級プログラマーへの第1歩 ～クラスプログラミング～



# メソッドの実装

## 🧩 クラスモジュール

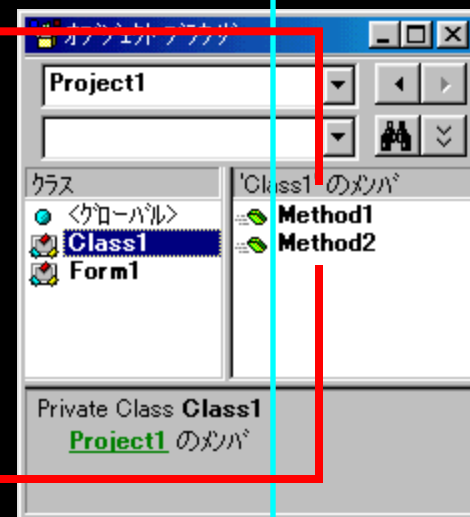
- (オブジェクト名) プロパティ=クラス名

## 🧩 メソッドの実装

- Publicな Sub または Function プロシージャ
- [挿入]メニューの[プロシージャ]

```
' // Class1 クラスモジュール  
  
' // 戻値のないメソッド  
Public Sub Method1(Optional pram1 As Long)  
  
End Sub  
  
' // 戻値のあるメソッド  
Public Function Method2() As Long  
  
End Function
```

```
' // UserForm1モジュール  
Dim n As Long  
Dim o As Class1  
Set o = New Class1  
  
o.Method1 12345  
n = o.Method2  
Set o = Nothing
```



# プロパティの実装

## プロパティの実装

- 1. Public変数を使う
- 2. Propertyプロシージャを定義

```
' // Class1 クラスモジュール
```

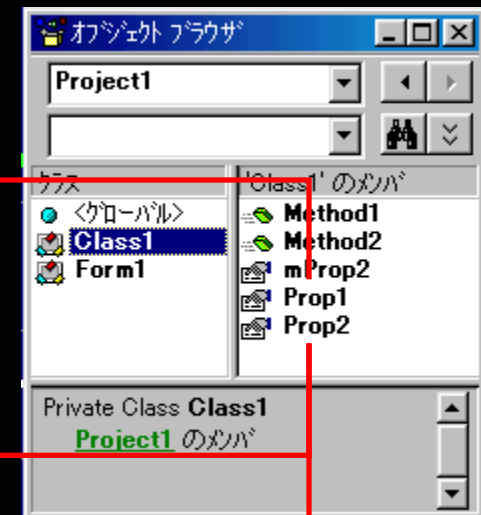
```
' // 1. パブリック変数によるプロパティ  
Public Prop1 As String
```

```
' // 2. プロパティ プロシージャによるプロパティ  
Private mProp2 As String
```

```
Public Property Get Prop2() As String  
    Prop2 = mProp2  
End Property
```

```
Public Property Let Prop2(ByVal NewValue As String)  
    mProp2 = NewValue  
End Property
```

```
' // UserForm1モジュール  
Dim s As String  
Dim o As Class1  
  
Set o = New Class1  
o.Prop1 = "ABC"  
o.Prop2 = "ABC"  
s = o.Prop2  
Set o = Nothing
```



# イベントの実装

## イベントの実装

- 定義: Public Event name [引数リスト]
- 発生: RaiseEvent name [引数リスト]

```
'// Class1 クラスモジュール
```

```
Public Event Event1 (ByVal param1 As Long, param2 As Boolean)
```

```
Public Sub Method1(Optional param1 As Long)
```

```
    Dim yn As Boolean
```

```
    RaiseEvent Event1(123, yn)
```

```
    If yn Then '.....
```

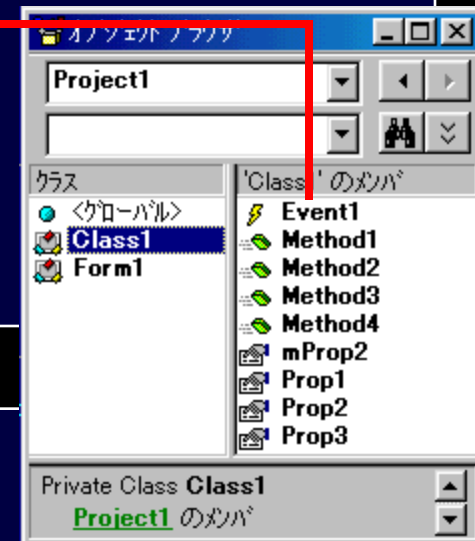
```
End Sub
```

```
'// UserForm1 フォームモジュール
```

```
Private WithEvents o As Class1
```

```
Private Sub o_Event1 (ByVal param1 As Long, param2 As Boolean)
```

```
End Sub
```



# Enumを使おう

```
'// ----- Class1 クラスモジュール -----
```

```
Private Enum apGender
```

```
    apMale = 1
```

```
    apFemale = 2
```

```
    apChild = 9
```

```
    [_apVIP] = -1
```

```
End Enum
```

**Enum**

```
Private lngGender As apGender
```

```
Public Property Get Gender() As apGender
```

```
    Gender = lngGender
```

```
End Property
```

```
Public Property Let Gender(ByVal NewGender As apGender)
```

```
    lngGender = NewGender
```

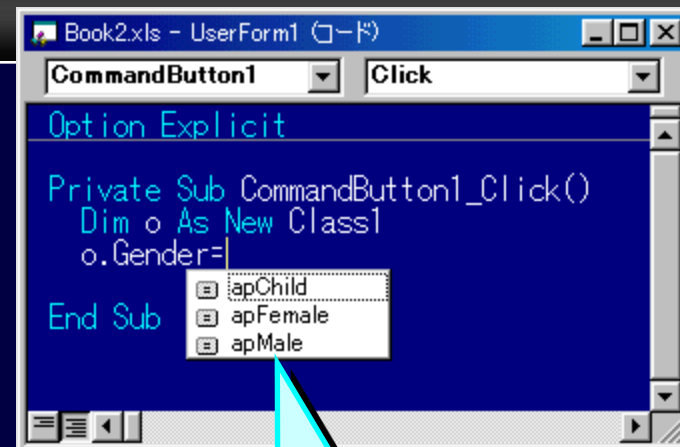
```
End Property
```

```
'// ----- UserForm1 モジュール -----
```

```
Private Sub CommandButton1_Click()
```

```
    o.Gender = apFemale
```

```
End Sub
```



**自動メンバ表示**

# コレクション クラス

## オブジェクトをコレクションとして管理するための 独自のクラス – 高度で安全な方法

- Step1. コレクション クラスモジュールの作成
- Step2. コレクションを保持するローカル変数
- Step3. メソッドの追加

```
' // クラスモジュール
```

```
Private mCol As Collection
```

```
Private Sub Class_Initialize()  
    Set mCol = New Collection  
End Sub
```

```
Private Sub Class_Terminate()  
    Set mCol = Nothing  
End Sub
```

```
Public Sub Add(Item As Fruit, _  
                Optional Key As String, _  
                Optional Before As Variant, _  
                Optional After As Variant)  
    mCol.Add Item, Key, Before, After  
End Sub
```

```
Public Function Item(Index As Variant) As Fruit  
    Set Item = mCol.Item(Index)  
End Function
```

```
Public Function Count() As Long  
    Count = mCol.Count  
End Function
```

```
Public Sub Remove(Index As Variant)  
    mCol.Remove Index  
End Sub
```

# VBAコレクション クラスの大惨事

🧩 VBAでは“プロシージャID”が設定できない！

🧩 規定値メンバが作れない

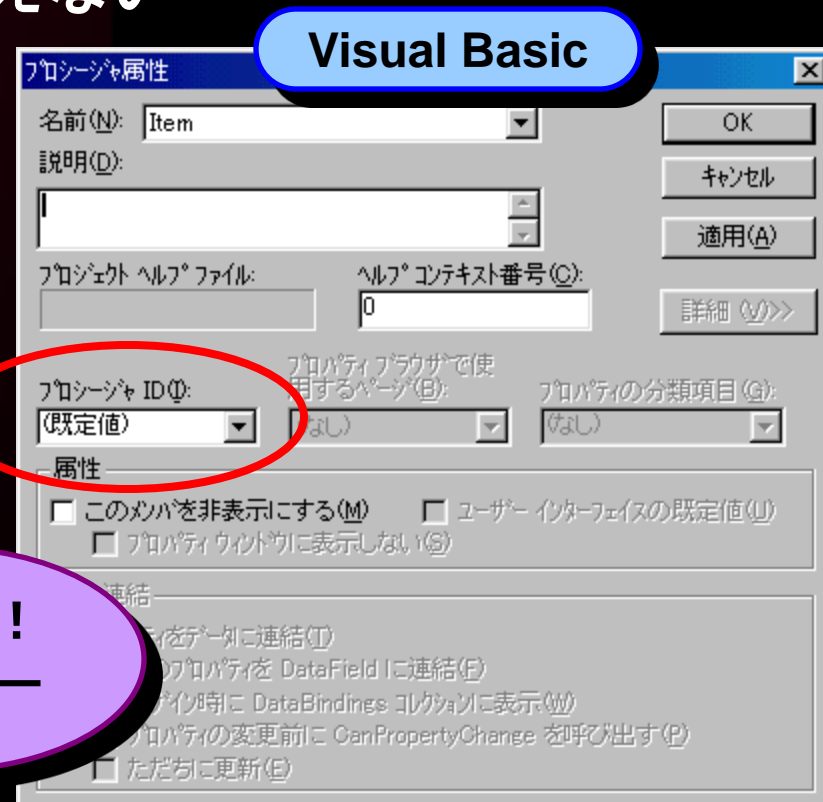
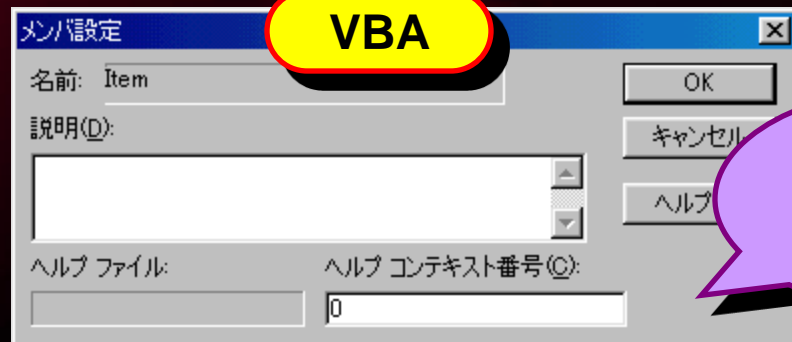
➤ Name, Item, Text, Caption, Value など

🧩 For Each ～ Next ループが使えない

➤ [\_NewEnum] メソッド

➤ VBAのコレクションはある

➤ For Each～Next は  
“列挙子”を使うため、  
DISP\_NEWENUM = -4  
のプロシージャを呼び出す



ゲゲッ！  
ないよー



# Office VBA Conference 2000

## パフォーマンス


お主は何でそんなに遅いんだ  
～パフォーマンスの向上のTips～

お主は何でそんなに遅いんだ ~パフォーマンスの向上のTips

# せっかちな貴方へ — 0.1秒でも速いコードを

 数値データ型変換(キャスト) Val関数は遅い

- CInt, CLng, CCur など Cxxx関数の方が速い

 文字列操作関数は"\$"マーク付の関数の方が速い

- Left - Left\$, Right - Right\$, Mid - Mid\$ など

 可能な限り"組み込み定数"を利用する

- vbCrLf, vbTab, vbKeyEscape, vbNullChar, vbRed など

 名前参照はわずかに遅い

- Sheets("Sheet1") v.s. Sheets(1)

 ループ中の DoEvents は時々発行する

- 長いループでの DoEventsの多用は CPUへ負荷を高めてしまう

 オブジェクトの参照は変数や With句を利用する

- x = Aaaaa. Bbbbb. Ccccc. Ddddd. Eeeee. Property
- Set objE = Aaaaa. Bbbbb. Ccccc. Ddddd. Eeeee  
x = objE. Property

お主は何でそんなに遅いんだ ~パフォーマンスの向上のTips

# 知らぬはほっとけ！

## 1行での複数条件文は注意すべし

- VBAはすべての条件を判断してから全体を評価している

```
If (n Mod 2) = 0 And foo(n) Then  
    ' // 該当処理  
    dummy = dummy + 1  
End If
```

```
If (n Mod 2) = 0 Then  
    If foo(n) Then  
        ' // 該当処理  
        dummy = dummy + 1  
    End If  
End If
```

## 文字列処理はまとめて行う

- 文字列管理を理解していれば...

```
Dim s As String  
For n = 1 To STEPS  
    s = s & "abc" & vbTab  
Next
```

```
Dim s As String  
Dim w As String  
For n = 1 To STEPS ￥ 100  
    w = vbNullString  
    For k = 1 To 100  
        w = w & "abc" & vbTab  
    Next  
    s = s & w  
Next
```

お主は何でそんなに遅いんだ ~パフォーマンスの向上のTips

# バインディングに注意すべし

## 🧩 Early (事前) バインディング & Late (実行時) バインディング

- Dim obj As Word.Application → Early バインディング
- Dim obj As Object → Late バインディング

## 🧩 Earlyバインディングの方が高速

- ただし、そのライブラリへの“参照設定”が必要

Early Bindling

Late Bindling

vTable

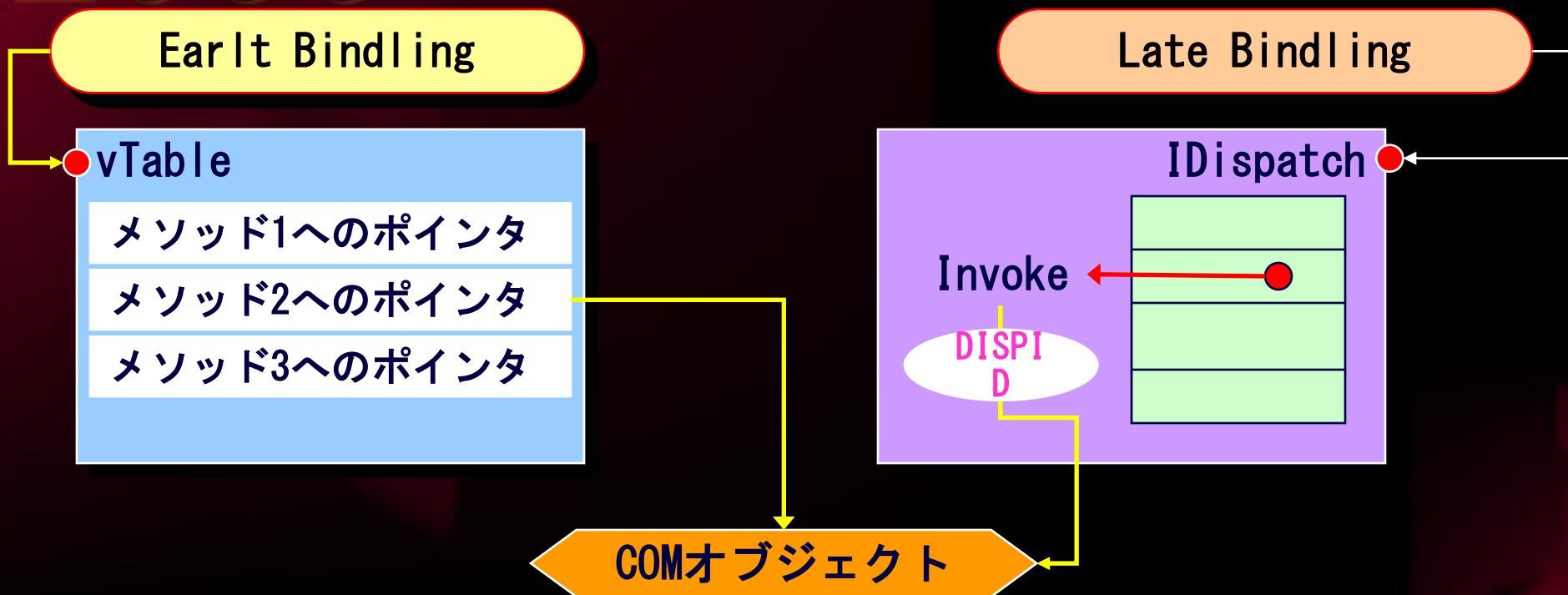
メソッド1へのポインタ  
メソッド2へのポインタ  
メソッド3へのポインタ

IDispatch

Invoke

DISPID

COMオブジェクト

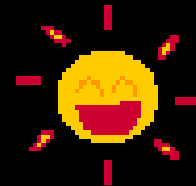


# Office VBA

# Conference

# 2000

**Thanks For Attending !**



General トラック 11:05～12:05

VBA ハイパーテクニク

Masato Hirai

**Office VBA**

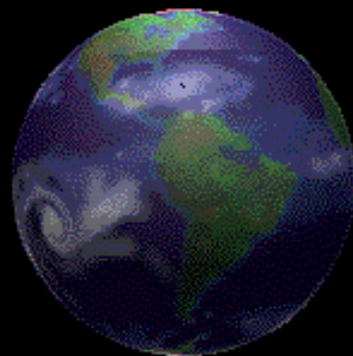
**Conference**

**2000**

**WHERE DO**

**YOU WANT TO**

**go**  
**TODAY?**



**Microsoft®**