単純明快！ オブジェクト指向超入門

オブジェクト指向のキーワード別に実践プログラミング

初音 玲
HATSUNE, Akira

Technology Tools

- ☑ Visual Basic .NET
- ❑ Visual C# .NET
- ❑ SQL Server 2000
- ❑ Oracle 9i
- ❑ Access 2002
- ❑ ASP.NET
- ❑ Internet Information Services
- ❑ Other:

Level

★★☆☆☆

Samples

・この記事で取り上げたソースコードおよびサンプルプログラムは、付録CD-ROMの¥DOTNET¥F01_01ディレクトリに収録しています。

¥VBPINHERIT1：クラスの使用例
¥VBPINHERIT2：継承の例
¥VBPINHERIT3：オーバーライドの例
¥VBPINHERIT4：抽象クラスの例
¥VBPINHERIT5：オーバーロードの例
¥VBPINHERIT6：ポリモーフィズムの例
¥VBPINHERIT7：ポリモーフィズムの例
¥COLUMN01：
コラム「OverridesとShadows」
¥COLUMN02：
コラム「メンバのスコープ」

はじめに

「オブジェクト指向プログラミング」というと聞き覚えのない言葉や、名前だけ知っているが意味がわからない言葉がいろいろでてきて、とにかく取っ付きにくいという印象がある。というのも、これまでは使いやすいオブジェクト指向言語が存在せず、まず理論から理解してゆかなければ、コンパイルすら通らなかったという背景もある。しかし、Visual Basic .NET（以下VB .NET）のように使いやすく、コードを読み解くのも簡単な言語が発売されたのだから、サンプルコードをたくさん読んで、コードからオブジェクト指向の意味を読み砕いてゆくことだってできるはずだ。

そこで、今回は、

「カプセル化」
「継承」
「派生クラス」
「抽象クラス」
「オーバーライド」
「オーバーロード」
「ポリモーフィズム」

というオブジェクト指向プログラミング独特の言葉と理論を、VB.NETのサンプルコードを動かしながら、手っ取り早く理解してしまおう。

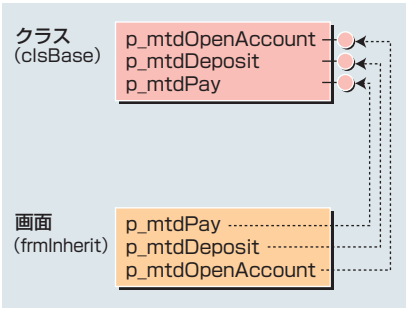
カプセル化（encapsulation）

オブジェクト指向プログラミングの基本のひとつめは“カプセル化”だ。

これは、情報とそれに対する操作手続きをひとつのまとまりとして定義して、内部の細かな仕様や構造を外から隠す、つまり「操作手続き以外に情報を操作できないようにする」という考え方だ。このまとまりをオブジェクト指向では「オブジェクト」と呼び、その設計図を「クラス」と呼ぶ。クラスに対する操作手続きにはメソッドやプロパティなどがあり、それらをまとめて、「クラスのメンバ」と呼ぶ。

VB.NETでもコードを記述するときには、クラスとして定義し、利用する

図1：継承



ときにはオブジェクト化（インスタンス化）して利用する（図1）。

クラスの定義

サンプルとしてリスト1を見てほしい。

このサンプルは、銀行の口座からの預金の引き出しをモデルにしたものだ。

預金を引き出すには「氏名」と「パスワード」、それにヒモ付けられた「口座番号」が必要になる。さらに預金残高を超えて引き出しができないように「残高」といった変数も必要だ。

では、リスト1を解説しよう。

定義1「Private」と宣言しているところは、カプセル化によってクラスの外には公開されない情報を記述している部分だ。Private属性でクラス定義の先頭部分に記述しているので、クラスの中からは共通情報として操作できるが、クラス外から操作できない。もちろん、Visual Studio .NETのIDEのインテリセンス機能（図2）でも表示されない。

定義2「Public」と宣言しているところは、隠蔽した情報に対する操作手

リスト1：クラスの例（サンプルvbpInherit1、clsBase.vb）

```
Public Class clsBase
  Private m_strNumber As String        ' 口座番号
  Private m_strName As String          ' 口座氏名
  Private m_decBalance As Decimal      ' 口座残高
  Private m_strPassword As String      ' パスワード

  Public _
  Sub p_mtdOpenAccount(ByVal vstrNumber As String, _
                       ByVal vstrName As String, _
                       ByVal vstrPassword As String)
    m_strNumber = vstrNumber
    m_strName = vstrName
    m_strPassword = vstrPassword
  End Sub

  Public _
  Sub p_mtdDeposit(ByVal vstrNumber As String, _
                   ByVal vdecAmount As Decimal)
    ' 入金する
    If vstrNumber = m_strNumber Then
      m_decBalance += vdecAmount
    Else
      Throw New OrignalErrorException( _
        "口座番号に誤りがあります。")
    End If
  End Sub

  Public _
  Sub p_mtdPay(ByVal vstrNumber As String, _
               ByVal vdecAmount As Decimal, _
               ByVal vstrPassword As String)
    ' 出金する
    If vstrNumber = m_strNumber Then
      If vstrPassword = m_strPassword Then
        m_decBalance -= vdecAmount
      Else
        Throw New OrignalErrorException( _
          "暗証番号に誤りがあります。")
      End If
    Else
      Throw New OrignalErrorException( _
        "口座番号に誤りがあります。")
    End If
  End Sub

  ReadOnly Property prpBalance() As Decimal
    ' 残高
    Get
      Return m_decBalance
    End Get
  End Property

  ReadOnly Property prpNumber() As String
    ' 口座番号
    Get
      Return m_strNumber
    End Get
  End Property

  ReadOnly Property prpName() As String
    ' 口座氏名
    Get
      Return m_strName
    End Get
  End Property
End Class

Public Class OrignalErrorException
  Inherits Exception
  Public Sub New(ByVal msg As String)
    MyBase.New(msg)
  End Sub
End Class
```