Programowanie Obiektowe w Visual Basic .NET 2005

Konrad Lipiński
07.04.2008

O czym będzie mowa

- **OOP a VB**
- **Klasy, interfejsy, struktury**
- **Operatory**
- **Zdarzenia**
- **Typy uogólnione**
- **Obsługa błędów**
- **Przestrzenie nazw**

OOP a VB

- **Klasy**
  - **Dane (`JobDescription`)**
  - **Metody (`AssignJob`)**
  - **Zdarzenia (`JobFinished`)**
- **Dziedziczenie (pojedyncze)**
- **Wielodziedziczenie po interfejsach (`Implements`)**
- **Polimorfizm**
- **Przeładowywanie (overloading)**

Klasy - deklaracja

```
   [attribute_list] _
   [Partial] _
   [accessibility] _
   [Shadows] _
   [inheritance] _
Class name[(Of type_list)]
   [Inherits parent_class]
   [Implements interface]
   members
End Class
```

Klasy - `attribute_list`

- **Opcjonalna lista atrybutów oddzielonych przecinkami**
- **Atrybut – dodatkowa informacja dla środowiska i/lub kompilatora**
- **Przykład:**
  - `Serializable`
- **Wyspecjalizowane właściwości**
  → odsyłamy do dokumentacji

Klasy - `Partial`

- **Opcjonalnie dzielimy treść klasy na kilka segmentów**
- **Przynajmniej jeden z nich ma być zadeklarowany przez `Partial`**

```
Partial Public Class Car
   Public HorsePower As Integer
End Class
..
Partial Public Class Car
   Public Price As Long
End Class
```

Klasy - `accessibility`

`Public`	**Wszyscy**
`Protected`	**Podklasy**
`Friend` (default)	**Projekt**
`Protected Friend`	**Podklasy w projekcie**
`Private`	**Tylko nasza klasa**

Klasy - **Shadows**

- **Klasa przesłania definicję z nadklasy**

```
Public Class Employee
  Public Class OfficeInfo
    Public Extension As String
  End Class
  Public Office As New OfficeInfo
End Class
Public Class Manager
  Inherits Employee
  Public Shadows Class OfficeInfo
    Public Extension As String
    Public SecretaryExtension As String
  End Class
  Public ManagerOffice As New OfficeInfo
End Class
```

Klasy - **`Shadows`** - c.d.

- **Employee zawiera 1 obiekt klasy `Office`**
- **Manager zawiera:**
  1 obiekt klasy `Office`
  1 obiekt klasy `ManagerOffice`

```
Dim emp As New Employee
Dim mgr As New Manager
emp.Office.Extension = "1111"
mgr.Office.Extension = "2222"
mgr.ManagerOffice.SecretaryExtension = "3333"
```

Metody a `Shadows` / `Overrides`

- **Definicja metody przesłania definicję z nadklasy, nawet jeśli sygnatura się nie zgadza**
- **Jeśli nadklasa definiuje kilka wersji metody, wszystkie zostają przesłonięte**
- **Alternatywa: `Overrides`**
  - **Przesłaniamy tylko tą metodę z nadklasy, która ma identyczną sygnaturę**
  - **Metoda `NotOverridable`: nie można użyć względem niej `Overrides`**

Klasy - `inheritance`

- **`NotInheritable`**
  nie można dziedziczyć z klasy
- **`MustInherit`**
  nie można instancjonować klasy

```
Public MustInherit Class Vehicle
  Public MustOverride Sub Drive()    ' pure virtual
End Class

Public Class Car
  Inherits Vehicle
  Public Overrides Sub Drive()
  End Sub
End Class
```

Klasy - **Implements** ***interface***

```
Public Interface Car
  Property HorsePower() As Integer
  Sub Sail()
End Interface

Public Interface Man
  Property Age() As Float
  Sub Talk()
End Interface

Public Class PanSamochodzik
  Implements Car
  Implements Man
  ..
End Class
```

Elementy statyczne klas - **Shared**

```
Public Class Student
  Shared NumStudents As Integer

  Public Shared Function NewStudent() As Student
    Dim new_student As New Student
    ' Add the new student to the database.
    NumStudents += 1
    Return new_student
  End Function
End Class

Dim student1 As Student = Student.NewStudent()
Dim student2 As Student = student1.NewStudent()
```

Klasy a struktury

- **Muszą zawierać co najmniej jedną zmienną lub zdarzenie**
- **Nie mogą dziedziczyć**
- **Klasy – referencja, struktury – bezpośrednio**

```
  [attribute_list] _
  [Partial] _
  [accessibility] _
  [Shadows] _
Structure name[(Of type_list)]
  [Implements interface]
  statements
End Structure
```

Przekazywanie parametrów

- **`ByRef` / `ByVal` (default)**
- **Przekazanie rwartości przez referencję powoduje utworzenie zmiennej tymczasowej i przekazanie referencji do niej**
- **Poniższy kod przejdzie bez efektu:**

```
Sub DoubleItByRef(ByRef X As Single)
  X*= 2
End Sub

DoubleItByRef(12)     ' Literal expression.
DoubleItByRef(X + Y)  ' Result of an expression.
```

Przekazywanie parametrów c.d.

- **Tak samo dzieje się przy pominięciu parametrów opcjonalnych (`Optional`)**

```
Sub UpdateEmployee(Optional ByRef emp_id As Integer = 0)
   ...
End Sub
```

- **Wzięcie wyrażenia w nawiasy oznacza wyliczenie wartości wyrażenia, więc..**

```
DoubleItByRef((value))
```

Boxing / Unboxing

- **Boxing – owijanie wartości w obiekt dowolna wartość może być traktowana jak obiekt, dalej przekazujemy referencję do tego obiektu**
- **Unboxing – odwijanie wartości z powrotem**
- **Wartość – typy wbudowane, struktury**
- **Przykład: dodanie struktury do kolekcji obiektów**

Instancjonowanie klas

```
Dim emp As Employee
emp = New Employee
Dim emp As New Employee

Public Class Person
  Public Name As String
  Public Sub New()
    Me.New("<unknown>")
  End Sub
  Public Sub New(ByVal name As String)
    Name = name
  End Sub
End Class

Dim wojtek As New Person("Wojtek")
Dim nieznanyZolnierz As New Person
```

Instancjonowanie klas c.d.

- **Jeśli zdefiniujemy jakikolwiek konstruktor, konstruktor bezargumentowy staje się nielegalny (o ile nie jest zdefiniowany)**

```
Public Class Person
  Public FirstName As String
  Public LastName As String
  Private Kwity As String = "na razie brak"
  Public Sub New( _
      Optional ByVal first_name As String = "<unknown>", _
      Optional ByVal last_name As String = "<unknown>")
    FirstName = first_name
    LastName = last_name
  End Sub
End Class
```

Instancjonowanie struktur

- **Struktura powstaje w momencie deklaracji**
- **Brak konstruktorów bezparametrowych**
- **Brak domyślnych wartości dla składowych**
- **Słówko `New` służy do wołania konstruktora**

```
Public Structure SPerson
  Public FirstName As String
  Public LastName As String
  Public Sub New(ByVal first_name As String, _
      Optional ByVal last_name As String = "<unknown>")
    FirstName = first_name
    LastName = last_name
  End Sub
End Structure
```

Instancjonowanie struktur c.d.

- **Inicjalizacja w momencie tworzenia**

```
Dim artist As New SPerson("Sergio", "Aragones")
```

- **Później reinicjalizacja**

```
Dim artist As SPerson
' Do something with artist.
...
' Reset FirstName and LastName to Nothing.
artist = New Sperson
...
' Set FirstName and LastName to Bill Amend.
artist = New SPerson("Bill", "Amend")
```

Konwersje typów

- **Jawna konwersja `CType(value, type)`**
- **Metoda `Parse(String)` dla wszystkich typów wbudowanych**
- **Niejawne konwersje poszerzające**
- **Niejawne konwersje zwężające (np. z `String` na `Integer`) dozwolone tylko przy wyłączonym `Option Strict`**

Option Explicit

- `On` **wymusza deklarowanie wszystkich zmiennych przed użyciem**
- `Off` **– każdy nowy symbol zostaje uzupełniony przez kompilator do deklaracji nowej zmiennej**
  - **Każda dedukowana zmienna jest przechowywana jako ogólny obiekt, żeby nie wiązać typu**
  - **Literówki**

Przeciążanie operatorów

```
Public Class Complex
  ..
  Public Shared Operator +(ByVal c1 As Complex, _
      ByVal c2 As Complex) As Complex
    Return New Complex(c1.Re + c2.Re, c1.Im + c2.Im)
  End Operator
  Public Shared Operator -(ByVal c1 As Complex) As Complex
    Return New Complex(c1.Im, c1.Re)
  End Operator
  Public Shared Narrowing Operator CType(ByVal c1 As Complex) _
      As Double
    Return System.Math.Sqrt(c1.Re * c1.Re + c1.Im * c1.Im)
  End Operator
  Public Shared Widening Operator CType(ByVal d1 As Double) _
      As Complex
    Return New Complex(d1, 0)
  End Operator
End Class
```

Operatory - ograniczenia

- **Pary operatorów – definiujemy albo oba albo żadnego (np. < i >)**
- **Klasa definujące operator musi pojawić się jako argument**
- **Klasa definująca operator konwersji `Ctype` musi pojawić się jako argument lub wynik**
- **Operator `Ctype` musi zawierać specyfikację `Widening` lub `Narrowing`**
- **Operatory `IsTrue` i `IsFalse` muszą dawać wynik typu `Boolean`**

Odśmiecanie: `Finalize`, `Dispose`

- **`Finalize`**
  - **Metoda dziedziczona z klasy `Object`**
  - **Wołana tuż przed odśmieceniem obiektu**
  - **Brak gwarancji na kolejność i moment wykonania**
- **`Dispose`**
  - **Opcjonalna, dziedziczona z interfejsu `IDisposable`**
  - **Wołana ręcznie natychmiastowo niszczy obiekt**
- **Obie metody są odpowiedzialne za zwolnienie zasobów specjalnych**

Odśmiecanie - przykład

```
Private Class Named
    Implements Idisposable

    Protected Overrides Sub Finalize()
        Dispose()
    End Sub

    Public Sub Dispose() Implements _
            System.IDisposable.Dispose
        Static done_before As Boolean = False
        If done_before Then Exit Sub
        done_before = True
        Debug.WriteLine("Freeing resources")
    End Sub

End Class
```

Dygresja - property procedures

- **Funkcje `Get` i/lub `Set` „definiujące" zmienną**
- **Dotyczy też definicji zwykłych zmiennych**
- **`ReadOnly` – tylko Get**
- **`WriteOnly` – tylko Set**

```
Public Property Name() As String
  Get
    Return m_FirstName & " " & m_LastName
  End Get
  Set(ByVal Value As String)
    m_FirstName = Value.Split(" "c)(0)
    m_LastName = Value.Split(" "c)(1)
  End Set
End Property
```

Zdarzenia

- **Obiekty wysyłają informacje do programu**
- **Instancja danej klasy używa `RaiseEvent` do poinformowania o zdarzeniu**
- **Zainstalowany (za pomocą `AddHandler`) handler obsługuje zdarzenie**

```
  [attribute_list] _
  [accessibility] _
  [Shadows] _                ' to replace a parent class event
Event event_name([handler_parameters]) _
  [Implements interface.event]
```

Zdarzenia - przykład

```
Public Interface IWorker
  Event WorkDone()
End Interface
Public Class SuperInstantWorker
  Implements IWorker
  Private Money As Integer = 0

  <Description("Occurs when a task is complete")> _
  Public Event WorkDone(ByVal task_number As Integer, _
      ByRef payment As Integer) _
    Implements IWorker.WorkDone

  Public Sub ScheduleWork(ByVal task_number As Integer)
    RaiseEvent(taks_number, Money)
  End Sub
End Class
```

Zdarzenia – przykład c.d.

```
Private Worker As SuperInstantWorker
Dim task_payment(997) As Integer = {1, 1, 2, 3, ..}

Private Sub HandleWorkDone(ByVal task_number As Integer, _
    ByRef payment As Integer)
  payment += task_payment(task_number)
End Sub

AddHandler Worker.WorkDone, AddressOf HandleWorkDone

For i As Integer = 0 To 997
  Worker.ScheduleWork(i)
Next i

RemoveHandler Worker.WorkDone, AddressOf HandleWorkDone
```

Zdarzenia – Custom Event

```
Public Class Employee
  Private m_Name As String
  Public Property Name() As String
    Get
      Return m_Name
    End Get
    Set(ByVal value As String)
      m_Name = value
      RaiseEvent NameChanged(Name)
    End Set
  End Property

  ' List to hold the event handler delegates.
  Private m_EventDelegates As New ArrayList

  ' Defines the event handler signature.
  Public Delegate Sub NameChangedDelegate(ByVal name As String)
```

Zdarzenia – **Custom Event** c.d.

```
  Public Custom Event NameChanged As NameChangedDelegate
    AddHandler(ByVal value As NameChangedDelegate)
      m_EventDelegates.Add(value)
    End AddHandler
    RemoveHandler(ByVal value As NameChangedDelegate)
      m_EventDelegates.Remove(value)
    End RemoveHandler

    RaiseEvent(ByVal name As String)
      Debug.WriteLine("RaiseEvent (" & name & ")")
      For Each deleg As NameChangedDelegate In m_EventDelegates
        deleg(name.Replace)
      Next deleg
    End RaiseEvent
  End Event
End Class
```

Zdarzenia – **Custom Event** c.d.

```
Public Sub NameChangedHandler(ByVal name As String)
  Debug.WriteLine("NameChangedHandler (" & name & ")")
End Sub

Dim emp As New Employee
AddHandler emp.NameChanged, AddressOf NameChangedHandler
AddHandler emp.NameChanged, AddressOf NameChangedHandler
emp.Name = "Fred"
RemoveHandler emp.NameChanged, AddressOf NameChangedHandler
emp.Name = "Flinston"
RemoveHandler emp.NameChanged, AddressOf NameChangedHandler

' Output:
RaiseEvent (Fred)
NameChangedHandler (Fred)
NameChangedHandler (Fred)
RaiseEvent (Flinston)
NameChangedHandler (Flinston)
```

Typy uogólnione – `Of` ***`type_list`***

```
Public Class Tree(Of data_type)
  Public RootObject As data_type
  ...
End Class

Dim my_tree As Tree(Of Employee)
my_tree = New Tree(Of Employee)
my_tree.RootObject = New Employee


Public Class Dict(Of key_type, value_type)

Dim emp_by_name As Dict(Of String, Employee)
```

Typy uogólnione – więzy

- **Można wyspecyfikować wymagania:**
  - **Jedną klasę z której trzeba dziedziczyć**
  - **Dowolną liczbę interfejsów**
  - **Obecność bezparametrowego konstruktora (słowo kluczowe `New`)**

```
Public Class StrangeGeneric( _
  Of Type1 As {IComparable, New}, _
  Type2, _
  Type3 As Control _
)
...
End Class
```

Obsługa błędów w VB

- **`#If DEBUG Then ... #End If`**
- **`Debug.Assert( _`**
  `the_order.Items IsNot Nothing, _`
  `"No items in order")`
- **Wyjątki**
- **Klasyka VB:** **`On Error GoTo line`**

Wyjątki

```
Try
  try_statements...
[Catch ex As exception_type_1
  exception_statements_1...]
[Catch ex As exception_type_2
  exception_statements_2...]
...
[Catch
  final_exception_statements...]
[Finally
  finally_statements...]    ' called when no Catch matches
End Try

Dim ex As New ArgumentException("Age must be nonnegative")
Throw ex
```

Wyjątki – klasa `Exception`

- **`InnerException`**
  Poprzedni wyjątek w łańcuchu
- **`Message` – krótki opis wyjątku**
- **`Source`**
  Nazwa aplikacji/obiektu zgłaszającego wyjątek
- **`StackTrace`**
  Zrzut stosu w momencie zgłoszenia wyjątku
- **`TargetSite`**
  Nazwa metody zgłąszającej wyjątek
- **`ToString`**
  Tekstowy opis wyjątku + stack trace

Klasyka VB – `On Error`

- **`On Error GoTo line`**
- **`On Error Resume Next`**
- **`On Error GoTo 0`**
- **`On Error GoTo -1`**
- **Tryb obsługi błędów**
- **Specjalny obiekt `Err` zawierający informacje o błędzie**

On Error GoTo line

- **Po wykonaniu tej instrukcji, VB zapamiętuje adres `line`**
- **Przy napotkaniu błędu wchodzi w tryb obsługi błędów i skacze pod adres `line`**

```
Private Sub ProcessPayroll()
  On Error GoTo LoadPayrollError
  LoadPayrollFile()
  Exit Sub

LoadPayrollError:
  MessageBox.Show("Error loading the payroll file.")

End Sub
```

On Error Resume Next

- **Po wykonaniu tej instrukcji wszystkie późniejsze błędne instrukcje są pomijane, a informacja o błędzie trafia do obiektu `Err`**

```
On Error Resume Next
X = CalculateValue()

Select Case Err.Number
  Case 0      ' No error. Do nothing.
  Case 11     ' Divide by zero. Set a default value.
    X = 1000
  Case Else
    MessageBox.Show("Error calc[X]." & Err.Description)
    Exit Sub
End Select
```

`On Error GoTo 0/-1` i inne

- **`On Error GoTo 0` wyłącza dowolny aktywny handler błędu (adres `line`)**
- **`On Error GoTo -1` – jw.**
  + dodatkowo kończy tryb obsługi błędów
- **`Resume` kończy tryb obsługi błędów i wraca do instrukcji, która spowodowała błąd**
- **`Resume Next` – jw, ale przechodzi do instrukcji następnej po tej, która spowodowała błąd**

Obiekt `Err`

- **`Err.Raise(5)` generuje błąd o numerze 5 ("Procedure call or argument is invalid")**
- **`Err.GetException()`**
  `daje` wyjątek odpowiadający błędowi zapisanemu w obiekcie `Err`

Przestrzenie nazw (namespace)

- **Typowa realizacja przestrzeni nazw z możliwością importowania części**
- **Możliwość tworzenia aliasów – skrótowych nazw dla przestrzeni nazw**

```
Namespace SchedulingClasses
   Public Class TimeSlot
      ..
   End Class
   ..
End Namespace

Imports [alias =] namespace[.element]
```

Dziękuję i zapraszam do dyskusji