

Języki Programowania na Platformie .NET

część 2

Dariusz Dereniowski

<http://www.kaims.pl/deren/net/>
deren@eti.pg.gda.pl

konsultacje: poniedziałek 10-11, oraz 17-18.

- koniec linii = koniec instrukcji
- znak _ oznacza kontynuację linii, np.:

```
Sub Print( \_  
           x As Integer )
```

Deklaracja zmiennej

```
[<lista_atr>] [dostępność] [[ Shared ] [ Shadows ] | [ Stat  
Dim [ WithEvents ] lista_zmiennych
```

- dostępność –
Public, Protected, Friend, Protected Friend, Private
- Shared – wszystkie instancje struktury lub klasy współdzielą tą samą zmienną
- Shadows – zmienna “ukrywa” zmienną o tej samej nazwie w klasie bazowej
- Static – zmienna istnieje pomiędzy wywołaniami procedury
- Dim – słowo kluczowe informujące o deklaracji zmiennej
- WithEvents – informuje, że zmienna jest obiektem typu, które może generować zdarzenia, które można przechwycić

Przykłady deklaracji

```
Dim napis As String = "Hello world."  
Dim tablica(), liczba As Integer
```

Gdy użyjemy Public, Protected, Friend, Protected Friend, Private, Shared, Shadows, Static, ReadOnly, lub WithEvents, to możemy pominąć dim

```
Dim a, b, c As Single, d, e, f As Double, g, h As Integer
```

Tablice

Deklaracja tablicy: `Dim a(), b(), c(,,) As Integer`

`Dim c(15) As String`

`Dim d(20,3) As Integer`

`Dim d(,) As Integer`

`d = New Integer(20,3) {}`

Tablicę `c` indeksujemy `0, ..., 15`.

Pusta tablica (jeden z wymiarów `-1`): `Dim tab(-1,5) As String`

Zmiana rozmiaru: `ReDim c(25)`

Odwołanie `d(5,1) = 3`

Inicjalizacja: `Dim e = New Integer() {1, 2, 3, 4}`

Deklaracja typu wyliczeniowego

```
[ <atrybuty> ] [ dostęp ] [ Shadows ]  
Enum nazwa [ As typ ]  
    składowe  
End Enum
```

- dostęp: Public, Protected, ...
- przy braku inicjalizacji wartość jest równa 0 dla pierwszej składowej lub jest o jeden większa od poprzednika na liście
- każda składowa ma postać:
[<atrybuty>] nazwa [= inicjalizacja]

Wybrane typy danych

Typ	Rozmiar	Wartości
Boolean	zal. od platf.	True lub False
Byte	1	0..255
SByte	1	-128..127
Char	2	0..65535
Short	2	-32768..32767
Integer	4	$-2^{31}..2^{31} - 1$
Long	8	$-2^{63}..2^{63} - 1$
Single	4	zmiennoprzec.
Double	8	zmiennoprzec.
Decimal	16	
String	zal. od platf.	
Object	4	wsk. każdy typ danych
UInteger	4	0..4294967295
ULong	8	0..(1.8...E+19)
UShort	2	0..65535

Funkcje konwersji

Funkcja	Konwersja do	Funkcja	Konwersja do
CBool	Boolean	CObj	Object
CByte	Byte	CByte	SByte
CChar	Char	CShort	Short
CDate	Date	CSgn	Single
Cdbl	Double	CStr	String
CDec	Decimal	CUInt	UInteger
CInt	Integer	CULng	ULong
CLng	Long	CUShort	Ushort

Wybrane instrukcje sterujące

```
If ... Then
```

```
...
```

```
End If
```

```
If ... Then
```

```
...
```

```
[ElseIf ... Then
```

```
...]
```

```
Else
```

```
...
```

```
End If
```

```
While ...
```

```
...
```

```
End While
```

```
For i=0 To 10 Step 1
```

```
...
```

```
Next [i]
```

```
Select x
```

```
Case 1
```

```
...
```

```
Case 2,3
```

```
...
```

```
Case Is < 20
```

```
...
```

```
Case Else
```

```
...
```

```
End Select
```

Wybrane operatory

Is – czy dwie referencje wskazują na ten sam obiekt.

```
if a Is b Then ...
```

IsNot – negacja powyższego.

TypeOf ... Is ... – czy obiekt jest określonego typu.

```
If TypeOf a Is String Then
```

Operator Like

```
Dim a As String
```

```
...
```

```
If a Like "?*@?*.?*" Then
```

```
...
```

Znak(i)	Znaczenie
---------	-----------

?	pasuje do pojedynczego znaku
---	------------------------------

*	zero lub więcej znaków
---	------------------------

#	pojedyncza cyfra
---	------------------

[...]	pasuje do dowolnego znaku podanego w nawiasach, np. [a-z]
-------	---

[!...]	pasuje do dowolnego znaku nie wyst. na liście
--------	---

A-Z	użyte w nawiasach pasuje do każdego znaku z zakresu A..Z
-----	--

Deklaracja procedur i funkcji

```
[ <atrybuty> ] [ Partial ] [ dostęp ] [ modyfikatory ]  
[ Shared ] [ Shadows ]  
Sub nazwa [(Of lista_typów_parametrów)] [(lista parametrów)]  
[ Implements lista_implementation | Handles lista_zdarzeń ]  
  [ wyrażenia ]  
  [ Exit Sub ]  
  [ wyrażenia ]  
End Sub
```

- dostęp: Public, Protected, ...
- modyfikatory: Overloads, Overrides, Overridable, NotOverridable, MustOverride
- lista typów parametrów: dla procedur generycznych (generic)
- element listy implementacji ma postać `interfejs.nazwa_proc`
- element listy zdarzeń ma postać: `obiekt.zdarzenie`

Przekazywanie parametrów

Przez wartość – procedura lub funkcja otrzymuje kopię zmiennej.

```
Public Sub A(ByVal i As Integer)
    i += 1 ' nie zmienia wart.
           ' org. zmiennej
End Sub
```

Przez referencję – procedura lub funkcja otrzymuje wskaźnik do zmiennej.

```
Public Sub A(ByRef i As Integer)
    i += 1 ' inkrementacja
           ' org. zmiennej
End Sub
```

Przekazywanie parametrów

```
[ <atrybuty> ] [ Optional ] [{ ByVal | ByRef }] [ ParamArray ]  
nazwa_parametru[( )] [ As typ_parametru ] [ = wartosc_domyslna ]
```

- jeśli parametr jest `Optional`, to wszystkie pozostałe też muszą takie być,
- parametr `Optional` musi mieć podaną wartość domyślną
- `ParamArray` jest używany w połączeniu z `ByVal`
- nie można używać `Optional` i `ParamArray` jednocześnie
- `ByVal` jest domyślnym mechanizmem; jeśli parametr jest referencją, to procedura może modyfikować wskazywany obiekt (choć nie może zmienić wartości samej referencji)

Definicja klasy

```
[ <atrybuty> ] [ dostęp ] [ Shadows ] _  
[ MustInherit | NotInheritable ] [ Partial ] _  
Class nazwa [ ( Of lista_typów ) ]  
    [ Inherits nazwa_klasy ]  
    [ Implements nazwa_interfejsów ]  
    [ zawartość ]  
End Class
```

- można definiować klasy zagnieżdżone

Słowo kluczowe Partial

- pozwala umieścić fragmenty kodu klasy w różnych fragmentach pliku lub w różnych modułach,
- co najmniej jeden z fragmentów musi być poprzedzony słowem `Partial`, ale dla przejrzystości kodu sugeruje się używać przy każdym fragmencie
- przydatne, gdy kilka osób pracuje nad różnymi fragmentami jednej klasy.

```
Partial Public Class A
    Public x As Integer
End Class
```

... inny kod ...

```
Partial Public Class A
    Public y As Double
End Class
```

Częściowe klasy i procedury

- częściowa procedura pozwala na oddzielenie jej deklaracji od implementacji
- używamy wraz z częściową klasą: w jednej części klasy umieszczamy deklarację, natomiast w drugiej implementację

```
Partial Class A
    Private x As Integer
    ' deklaracja:
    Partial Private Sub powieksz()
    End Sub
End Class

...

Partial Class Product
    ' implementacja:
    Private Sub powieksz()
        x = x+1
    End Sub
End Class
```

Dostępność

- `Public` – klasa dostępna dla całego kodu, również poza modułem
- `Protected` – stosowane tylko w odniesieniu do klas; dostęp tylko w obrębie klasy i klas pochodnych
- `Friend` (domyślne) – klasa dostępna dla całego kodu w module oraz poza modułem wewnątrz tego samego projektu. Różnice w stosunku do `Public` są przy tworzeniu bibliotek.
- `Protected Friend` – suma dwóch powyższych.
- `Private` – klasa dostępna tylko wewnątrz modułu.

Dziedziczenie

- `MustInherit` – nie można tworzyć instancji takiej klasy (klasa abstrakcyjna).
- `NotInheritable` – nie można dziedziczyć.

```
Public MustInherit Class Figura
    Public MustOverride Sub Pole()
End Class
```

```
Public Class Kwadrat
    Inherits Figura
    Public Overrides Sub Pole()
        ...
    End Sub
End Class
```

Dziedziczenie – przykład

```
Private MustInherit Class A
```

```
    Public x As Integer
```

```
    Public Sub f(ByVal x As Integer)
```

```
        Me.x = x
```

```
    End Sub
```

```
End Class
```

```
Private Class B
```

```
    Inherits A
```

```
    Public Overloads Sub f(ByVal x As Integer)
```

```
        Me.x = 2 * x
```

```
    End Sub
```

```
End Class
```

```
Dim a As A
```

```
a = New B
```

```
a.f(5) ' a.x = 5
```

```
Dim b As B
```

```
b = New B
```

```
b.f(5) ' b.x = 10
```

Dziedziczenie – przykład

```
Private MustInherit Class A
    Public x As Integer
    Public Sub New()
        x = 1
    End Sub
End Class
```

```
Private Class B
    Inherits A
    Public Shadows x As Integer
    Public Sub New()
        x = 2
    End Sub
End Class
```

```
Dim a As A
Dim b As B
a = New B
b = New B
Console.WriteLine(a.x) '1
Console.WriteLine(b.x) '2
```

Tworzenie obiektu

- deklaracja zmiennej dowolnej klasy to alokacja 4 bajtów pamięci i przypisanie jej domyślnej wartości `Nothing`
- przy braku konstruktorów, przy tworzeniu obiektu używamy słowa `New` bez parametrów
- konstruktor to procedura o nazwie `New` (zwykle kilka i z parametrami).
- pierwsza linia konstruktora powinna być wywołaniem konstruktora w klasie bazowej, co realizujemy poprzez `MyBase.New(parametry)`
- powyższe można pominąć jeśli klasa bazowa posiada konstruktor bez parametrów i wówczas zostanie on wywołany domyślnie

Konstruktory – przykład

```
Public Class Osoba
    Public imie As String
    Public nazw As String

    Public Sub New(ByVal i As String, ByVal n As String)
        imie = i
        nazw = n
    End Sub

    Public Sub New()
        Me.New("-", "-")
    End Sub
End Class
```

Tworzenie obiektu – przykład

```
Public Class Osoba
    Public imie As String
    Public nazw As String

    Public Sub New(Optional ByVal i As String = "-", _
                   Optional ByVal n As String = "=")
        imie = i
        nazw = n
    End Sub
End Class

Dim a As Osoba = New Osoba           ' -,=
Dim b As Osoba = New Osoba("Jan")   ' Jan,=
Dim c As Osoba = New Osoba("Jan", "K") ' Jan,K
```

Uwaga o składowych Private

Jeśli składowa klasy (np. zmienna) jest zadeklarowana jako Private, to obiekt danej klasy ma również dostęp do tej składowej w innej instancji tej samej klasy.

```
Public Class Punkt
    Private x As Double

    Public Function Porownaj(ByVal inny_punkt As Punkt) As Boolean
        If x > inny_punkt.x Then
            return True
        Else
            return False
        End If
    End Function
End Class
```

Metoda Finalize

- *Garbage collector* zwalnia pamięć przydzieloną obiektom, do których nie ma referencji w programie. Możemy zlecić wykonanie kodu tuż przed zwolnieniem pamięci przesłaniając metodę `Finalize` zdefiniowaną w `Object`.
- metoda `Finalize` nie musi być wywołana bezpośrednio po tym, gdy przestaną istnieć w programie referencje do obiektu
- w celu natychmiastowego zwalniania zasobów używamy `Dispose`

```
Public Class A
    ...
    Protected Overrides Sub Finalize()
        ...
    End Sub
End Class
```

Przeciążanie operatorów

- można przeciążać następujące operatory: `+`, `-`, `*`, `/`, `\`, `^`, `&`, `<<`, `>>`, `=`, `<>`, `<`, `>`, `<=`, `>=`, `Mod`, `Not`, `And`, `Or`, `Xor`, `Like`, `IsTrue`, `IsFalse`, `CType`
- pewne operatory tworzą pary – trzeba przeciążać również drugi. Pary: `=`, `<>`; `<`, `>`; `<=`, `>=` oraz `IsTrue`, `IsFalse`.
- nie można przeciążać:
`AndAlso`, `OrElse`, `New`, `TypeOf` ... `Is`, `Is`, `IsNot`,
`AddressOf`, `GetType`, `AsType`
- definiujemy podobnie jak procedure/funkcje, przy czym `Sub/Function` używamy słowa kluczowego `Operator`, a nazwą metody jest przeciążany operator

Przeciążanie operatorów

```
Class Punkt
```

```
    Private x, y As Double
```

```
    Public Sub New(ByVal x As Double, ByVal y As Double)
```

```
        Me.x = x
```

```
        Me.y = y
```

```
    End Sub
```

```
    Public Shared Operator +(ByVal a As Punkt, ByVal b As Punkt)
```

```
        Dim r As New Punkt( a.x + b.x, a.y + b.y )
```

```
        Return r
```

```
    End Operator
```

```
End Class
```

Przeciążanie operatorów – przykład

```
Private Class A
    Public x As Integer
    Public Sub New()
        x = 1
    End Sub

    Public Shared Widening Operator CType(ByVal x As Integer) As A
        Dim r As A = New A
        r.x = x
        Return r
    End Operator
End Class

...
Dim a As A = New A
a = 101
```

Uwaga: musimy uzyc:

- Narrowing – gdy możliwy jest błąd rzutowania (zgłosimy wyjątek)
- Widening – gdy rzutowanie zawsze się powiedzie

Properties

```
Private Class B
    Private x As Integer

    Public Property x_pr()
        Get
            Return x
        End Get
        Set(ByVal value)
            x = value
        End Set
    End Property
End Class
```

Obsługa błędów

```
Try
    [ kod ]
    [ Exit Try ]
[ Catch [ wyjątek [ As typ ] ] [ When wyrażenie ]
    [ kod ]
    [ Exit Try ] ]
[ Catch ... ]
[ Finally
    [ kod ] ]
End Try
```

Obsługa błędów

- gdy pojawi się błąd podczas wykonywania bloku `Try`, poszukiwany jest blok `Catch`, którego typ wyjątku 'pasuje' do błędu
- kod w bloku `Finally` wykonywany jest po tym, gdy kod bloku `try` wykonał się bezbłędnie lub gdy zakończy się obsługa błędu w bloku `Catch`
- wyjątki od powyższego, to napotkanie wyrażenia `End` lub pojawił się wyjątek `StackOverflowException` w bloku `Try` lub `Catch`

- nie ma możliwości wielokrotnego dziedziczenia w VB,
- klasa może implementować wiele interfejsów.

```
[ <atrybuty> ] [ dostęp ] [ Shadows ] _  
Interface nazwa [ ( Of lista_typów ) ]  
    ...  
End Interface
```

```
Public Interface IFigura  
    Function pole() As Double  
    Property X() As Double  
    Property Y() As Double  
End Interface
```

Interfejsy

```
Public Class Kwadrat
    Implements IFigura

    Public Function pole() As Double Implements IFigura.pole
    End Function

    Property X() As Double Implements IFigura.X
        Get
        End Get
        Set(ByVal a As Double)
        End Set
    End Property

    ...
End Class
```

Struktury

```
[ <atrybuty> ] [ dostęp ] [ Shadows ] [ Partial ] _  
Structure nazwa [ ( Of lista_typów ) ]  
    [ Implements nazwy_interfejsów ]  
    ...  
End Structure
```

Różnice w stosunku do klasy:

- struktury nie mogą dziedziczyć,
- zmienna będąca strukturą przechowuje całą zawartość struktury (value type), natomiast zmienna obiektowa jest referencją (4 bajty).
- przypisanie w przypadku struktury kopiuje całą jej zawartość; w przypadku obiektów - przypisanie powoduje, że obie zmienne wskazują ten sam obiekt.

Generics – deklaracje generyczne

```
Public Class Lista(Of t [As IComparable])  
    ...  
End Class  
...  
Dim l As Lista(Of String)
```

- kontrola typów na etapie kompilacji

Przestrzenie nazw (Namespaces)

- przestrzenie nazw pozwalają rozwiązywać problemy związane z jednakowymi nazwami, np. klas.
- przestrzenie nazw mogą tworzyć hierarchiczną strukturę
- nie są typami, służą do organizacji kodu

```
Dim a As Siec.Polaczenie
```

```
Dim b As Zasoby.Polaczenie
```

Namespaces – Imports

- wyrażenia typu `System.Drawing.Rectangle` nie są wygodne i utrudniają czytanie kodu.
- umieszczamy na początku pliku `Imports System.Drawing` i wówczas w kodzie wcześniejsze wyrażenie zastępujemy `Rectangle`
- plik może zawierać (zawsze podane na początku) dowolną liczbę instrukcji `Imports`
- jeśli chcemy importować różne moduły dla różnych fragmentów kodu, to trzeba kod podzielić pomiędzy kilka plików.

Namespaces – aliasy

Pełna składnia: Imports [alias =] namespace[.element] Np:

```
Imports D2 = System.Drawing.Drawing2D
```

```
...
```

```
Dim dash_style As D2.DashStyle = D2.DashStyle.DashDotDot
```

- Jeśli zdefiniowaliśmy alias, to *trzeba* go użyć, aby się odwołać do elementu przestrzeni nazw.
- aliasy są potrzebne, gdy importujemy dwie przestrzenie nazw, które zawierają definicje jednakowo nazwanych elementów (np. klas)

Namespaces – importowanie elementu

Oprócz importowania przestrzeni nazw można importować jej element.

```
Imports AliasDoListBox = ListBoxProject.Form1.ListBox  
...  
Dim x As AliasDoListBox
```

Namespaces – definiowanie własnych

- Każdy projekt posiada swoją przestrzeń nazw – każdy element w nim zdefiniowany do niej należy (bezpośrednio lub pośrednio).
- Można definiować własną przestrzeń nazw (wewnątrz przestrzeni nazw projektu):

```
Namespace StrukturyDanych
  Public Class Lista
    ...
  End Class
End Namespace
```

Namespaces – definiowanie własnych

- Do powyżej zdefiniowanej klasy `Lista` kod wewnątrz przestrzeni `StrukturyDanych` odwołuje się poprzez samą nazwę `Lista`
- Kod poza tą przestrzenią odwołuje się następująco:

```
Dim l As New MyApplication.StrukturyDanych.Lista
```

zakładając, że `MyApplication` jest przestrzenią nazw projektu.

Namespaces – definiowanie własnych

- wewnątrz przestrzeni nazw można definiować inne przestrzenie nazw, klasy, struktury, moduły, typy wyliczeniowe, interfejsy.
- wewnątrz przestrzeni nazw nie można definiować bezpośrednio zmiennych, właściwości (properties), procedur, zdarzeń (events).
- można jedną przestrzeń nazw definiować w kilku fragmentach. Przydatne, gdy:
 - chcemy definicje klas umieszczać w osobnych plikach,
 - różne osoby pracują nad różnymi fragmentami kodu,
 - chcemy rozszerzyć istniejące przestrzenie nazw.

Typy CLS

Typ .Net	Typ VB	C#	zakres
System.Byte	Byte	byte	0..255
System.SByte	SByte	sbyte	-128..127
System.Int16	Short	short	-32 768..32 767
System.UInt16	UShort	ushort	0..65 535
System.Int32	Integer	int	$-2^{31}..2^{31} - 1$
System.UInt32	UInteger	uint	$0..2^{32} - 1$
System.Int64	Long	long	$-2^{63}..2^{63} - 1$
System.UInt64	ULong	ulong	$0..2^{64} - 1$
System.Single	Single	float	32 bity
System.Double	Double	double	64 bity
System.Decimal	Decimal	decimal	128 bitów
System.Char	Char	char	unicode
System.String	String	string	16· dł.
System.Boolean	Boolean	bool	True,False
System.Object	Object	object	32 bity