

# Języki Programowania na Platformie .NET

## Reflection

- Reflection wykorzystuje głównie klasy znajdujące się w przestrzeni nazw `System.Reflection`
- wyjątkiem jest klasa `System.Type`, która reprezentuje typ w module zarządzanego kodu
- Możliwości
  - analiza zewnętrznego kodu podczas działania programu
  - 'late binding'

# Klasa Assembly

Klasa reprezentuje pojedynczy moduł (.NET assembly). Przykład uzyskania referencji do takiego obiektu:

```
' pobranie referencji do bieżącego modułu  
Dim a As Assembly = Assembly.GetExecutingAssembly()  
' pobranie referencji do modułu zawierającego określony typ  
a = Assembly.GetAssembly( GetType( DateTime ) )  
' pobranie z określonej lokalizacji  
a = Assembly.LoadFrom( "... " )  
' pobranie tylko do celów 'incpekcji'  
a = Assembly.ReflectionOnlyLoadFrom( "... " )
```

# Wykorzystanie obiektu Assembly

- uzyskanie kolekcji typów zdefiniowanych w module

```
For Each t As Type in a.GetTypes()  
    Console.WriteLine( t.FullName() )  
Next
```

- pobranie typu na podstawie jego nazwy

```
Dim t As Type = a.GetType( "... " )
```

- szczegółowe informacje o module

```
Dim an As AssemblyName = a.GetName()  
an.ProcessorArchitecture()  
an.VersionMajor()  
an.VersionMinor()
```

## Klasa Type – tworzenie obiektu

' na podstawie nazwy klasy:

```
Dim t As Type = GetType( String )
```

' na podstawie obiektu:

```
Dim i As Integer
```

```
t = i.GetType() ' System.Int32
```

' na podstawie nazwy klasy (string)

```
t = GetType( "System.DateTime" )
```

Jeśli funkcja `GetType` nie może odnaleźć wskazanego typu, to uruchamiane jest zdarzenie `TypeResolve` w klasie `AppDomain`. Pozwala to m.in. obsłużyć sytuacje wyjątkowe lub zmienić standardowy sposób nazywania typów.

## Wybrane metody klasy Type

```
If t.IsClass Then
    ...
ElseIf t.IsInterface Then
    ...
ElseIf t.IsEnum Then
    ...
ElseIf t.IsValueType Then
    ... ' Enum to również ValueType, więc niezbędna taka ko
Else
    ... ' sterowanie nigdy nie dotrze tutaj
EndIf
```

## Wybrane metody klasy Type

```
If t.IsPublic Then  
ElseIf t.IsNotPublic Then  
EndIf
```

```
If t.IsAbstract Then  
EndIf
```

```
If t.IsSealed Then  
EndIf
```

```
t.BaseType.FullName()  
If t.IsInstanceOfType( obj ) Then  
EndIf
```

```
If obj.GetType().IsSubclassOf( GetType( MojaKlasa ) ) Then  
EndIf
```

# Przetwarzanie elementów klasy

Następujący kod

```
Dim info() As MemberInfo = t.GetMembers()  
For Each i As MemberInfo in info  
    ...  
Next
```

przetwarza elementy wchodzące w skład klasy: pole, 'property', metoda, konstruktor, 'event', zagnieżdżona klasa. Można też przetwarzać składowe wybranego typu (`t.GetFields`, `GetProperties`, `GetMethods`, `GetEvents`, `GetConstructors`, `GetInterfaces`, `GetNestedTypes`, `GetDefaultMembers`)

## Uzyskiwanie referencji do wybranej metody

```
Dim mi As MethodInfo = t.GetMethod( "NazwaMetody" )
```

' jeśli metoda jest przeciążona, to trzeba wyspecyfikować  
' dokładną listę parametrów:

```
Dim args() As Type = { GetType( Integer ), GetType( Double ) }  
mi = GetMethod( "InnaNazwa", args )
```

' gdy parametry, to typy referencyjne lub tablica:

```
GetType( Double ).MakeRefType()  
GetType( Integer ).MakeArrayType( 4 )
```

# Dynamiczne tworzenie obiektu

' zwykle referencja jest typu Object (nie znamy typu na  
' etapie kompilacji)

```
Dim o As Object = Activator.CreateInstance( t )
```

' konstr. z parametrami: przekazujemy je jako tablicę:

```
Dim args() As Object = { 1, "Hello" }
```

```
o = Activator.CreateInstance( t, args )
```

' uzyskanie dostępu do pola:

```
Dim f As FieldInfo = t.GetField( "NazwaPola" )
```

```
f.GetValue( o ) ' lub f.SetValue( o, wartość )
```

' wywołanie metody:

```
Dim m As MethodInfo = t.GetMethod( "NazwaMetody" )
```

```
Dim args() As Object = { 4, "Hello" }
```

```
m.Invoke( o, args )
```