

IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK

DODAJ DO KOSZYKA

CENNIK I INFORMACJE

ZAMÓW INFORMACJE
O NOWOŚCIACH

ZAMÓW CENNIK

CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

Programowanie obiektowe w Visual Basic .NET dla każdego

Autorzy: Richard Simon, Leslie Koorhan, Ken Cox

Tłumaczenie: Jadwiga Gnybek

ISBN: 83-7197-783-2

Tytuł oryginału: [TY Object Oriented Programming
with Visual Basic.NET in 21 Days](#)

Format: B5, stron: 374



Visual Basic to język programowania pozwalający nawet początkującym programistom pisać zaawansowane aplikacje wykorzystujące wszystkie możliwości systemów operacyjnych z rodziny Windows. Jego kolejne wersje w coraz większym stopniu korzystały z technik programowania zorientowanego obiektowo (OOP). Najnowsza z nich, Visual Basic .NET jest uwieńczeniem tej ewolucji.

Jeśli chcesz tworzyć aplikacje w VB .NET, musisz nauczyć się programować obiektowo. Z pewnością pomoże Ci w tym książka „Programowanie obiektowe w Visual Basic .NET”. Informacje w niej zawarte przydadzą się także, gdy zaczniesz używać innych języków programowania opartych na obiektach, takich jak C++, C# czy Java.

Ten podręcznik, łączący praktyczne ćwiczenia z niezbędną dawką przydatnej teorii, nauczy Cię samodzielnie tworzyć aplikacje używające obiektów, dziedziczenia, polimorfizmu i sterowania za pomocą zdarzeń. Dowiesz się również jak obsługiwać błędy. Odrębne rozdziały poświęcono obiektom zdalnym (aplikacjom klient-serwer), udostępnianiu aplikacji poprzez Internet tworzeniu atrakcyjnego interfejsu użytkownika oraz językowi XML. Poznasz także środowisko projektowe Visual Studio .NET. Swoją wiedzę będziesz mógł sprawdzić za pomocą licznych quizów.

Książka zawiera:

- Wprowadzenie do programowania obiektowego w Visual Basic .NET
- Wyjaśnienie terminologii związanej z programowaniem obiektowym
- Omówienie środowiska Visual Studio .NET oraz tworzenia interfejsów użytkownika za pomocą Windows Forms i WebForms
- Praktyczne ćwiczenia i quizy sprawdzające wiedzę



Spis treści

O Autorze	11
Wstęp	13
Jak zorganizowana jest ta książka?	14
Konwencje użyte w książce	15
Rozdział 1. Programowanie zorientowane obiektowo — budowa aplikacji sterowanej zadaniami	17
Tworzenie projektu	18
Definiowanie obiektu SimpleMsg.....	19
Tworzenie obiektów w Visual Basic .NET.....	19
Dodawanie właściwości.....	20
Tworzenie klasy SmartMsg.....	22
Dziedziczenie w Visual Basic .NET.....	22
Dodanie metody Write()	23
Dodanie metody Read()	24
Budowa obiektu MsgReader	25
Tworzenie okna formularza	26
Wywołanie obiektu SmartMsg	28
Wyświetlanie obiektu SmartMsg.....	29
Obsługa zdarzeń OK i Anuluj.....	30
Łączymy wszystko w całość	31
Sub main().....	31
Ustawianie parametrów startowych.....	31
Podsumowanie	32
Pytania i odpowiedzi	32
Warsztaty.....	33
Quiz.....	33
Ćwiczenie.....	33
Rozdział 2. Terminologia wykorzystywana w OOP	35
Obiekt.....	35
Klasa.....	36
Pola i zakresy	37
Prywatne	38
Chronione.....	39
Publiczne.....	39
Pola statyczne.....	40
Klasy finalne i abstrakcyjne.....	41
Właściwości	43

Metody i ich działanie	44
Zdarzenia i delegaty	45
Konstruktory.....	47
Destruktry.....	48
Interfejsy.....	49
Podsumowanie	50
Pytania i odpowiedzi	50
Warsztaty.....	51
Quiz.....	51
Ćwiczenie.....	51
Rozdział 3. Zawieranie danych i cech w obiekcie	53
Projektowanie publicznego interfejsu klasy.....	54
Ochrona pól klas.....	55
Definiowanie prywatnych pól klasy.....	56
Zastosowanie hermetyzacji w projektowaniu obiektów	58
Podsumowanie	64
Pytania i odpowiedzi	64
Warsztaty.....	65
Quiz.....	65
Ćwiczenie.....	65
Rozdział 4. Tworzenie nowych obiektów na bazie obiektów już istniejących.....	67
Mechanizm dziedziczenia	68
Klasa bazowa	68
Dziedziczenie klas	69
Dodawanie danych.....	70
Dodawanie funkcji.....	70
Optymalizacja klas	70
Nadpisywanie — zmiana funkcjonalności istniejących metod	71
Wykorzystanie klas abstrakcyjnych.....	72
Wykorzystanie pól chronionych	73
Wykorzystanie pól współdzielonych	73
Tworzenie klas finalnych.....	74
Wykorzystanie dziedziczenia w Visual Basic .NET.....	75
Proste dziedziczenie klas — tworzenie klasy MailMsg	75
Dziedziczenie — tworzenie formularza transakcji	77
Podsumowanie	81
Pytania i odpowiedzi	82
Warsztaty.....	83
Quiz.....	83
Ćwiczenie.....	83
Rozdział 5. Nadawanie obiektom cech polimorficznych	85
Zalety polimorfizmu.....	85
Wykorzystanie polimorfizmu w Visual Basic .NET.....	86
Dziedziczenie polimorfizmu	86
Klasy abstrakcyjne a polimorfizm	90
Polimorfizm interfejsu	92
Podsumowanie	94
Pytania i odpowiedzi	95
Warsztaty.....	95
Quiz.....	95
Ćwiczenie.....	96

Rozdział 6. Budowa obiektów złożonych poprzez łączenie obiektów	97
Różnica pomiędzy kompozycją a dziedziczeniem	97
Asocjacja	99
Agregacja	99
Kompozycja	99
Projektowanie obiektów przy użyciu kompozycji	100
Podsumowanie	105
Pytania i odpowiedzi	105
Warsztaty	106
Quiz	106
Ćwiczenie	106
Rozdział 7. Poznajemy środowisko programistyczne Visual Basic .NET	107
Praca w Microsoft .NET Framework	108
Common Language Runtime (CLR)	108
Biblioteka klas .NET	109
Praca z Visual Basic .NET IDE	109
Wykorzystanie okna Solution Explorer	111
Project	112
Referencje	113
Referencje WWW	113
Toolbox	114
Form Designer	115
Uruchamianie i debugowanie aplikacji	117
Wykorzystywanie systemu pomocy Help Online	118
Podsumowanie	120
Pytania i odpowiedzi	120
Warsztaty	121
Quiz	121
Ćwiczenie	121
Rozdział 8. Typy danych w Visual Basic .NET	123
Poznajemy typy danych Visual Basic .NET	124
Typ Object	124
Synonimy typów Common Numeric Data	125
Używanie zmiennych typu łańcuch znaków (String)	126
Praca z danymi typu Data	127
Używanie zmiennych typu Boolean	128
Tworzenie struktur	128
Praca ze zmiennymi typu wyliczeniowego	129
Używanie tablic	131
Podsumowanie	132
Pytania i odpowiedzi	132
Warsztaty	133
Quiz	133
Ćwiczenie	133
Rozdział 9. Organizowanie klas w grupy	135
Przestrzenie nazw	136
Praca z przestrzeniami nazw .NET	137
Używanie przestrzeni nazw	138
Tworzenie przestrzeni nazw	140
Podsumowanie	141
Pytania i odpowiedzi	141
Warsztaty	142
Quiz	142
Ćwiczenie	142

Rozdział 10. Tworzenie interfejsu użytkownika	143
Wybieramy właściwy formularz	144
Używanie formularzy Windows.....	145
Formularze Windows.....	145
Tworzenie formularza Windows.....	149
Rozbudowa formularza Windows.....	152
Formularze WWW	153
Praca z formularzem WWW i ASP.NET.....	153
Tworzenie formularza WWW ASP.NET	154
Podsumowanie	158
Pytania i odpowiedzi.....	158
Warsztaty.....	159
Quiz.....	159
Ćwiczenie.....	159
Rozdział 11. Tworzenie i używanie komponentów	161
Tworzenie klas komponentów Visual Basic .NET	161
Komponenty.....	162
Projektowanie i implementacja komponentów	164
Programowanie klas do użycia w trakcie pracy aplikacji	166
Klasy Reflection.....	167
Programowanie obiektów nieznanych	168
Podsumowanie	170
Pytania i odpowiedzi.....	171
Warsztaty.....	171
Quiz.....	171
Ćwiczenie.....	171
Rozdział 12. Budowa aplikacji WWW	173
Zorientowane obiektowo aplikacje ASP.NET w Visual Basic .NET	173
Tworzenie projektu WWW.....	174
Tworzenie interfejsu użytkownika.....	174
Serwisy WWW.....	181
Tworzenie serwisów WWW	182
Używanie serwisów WWW.....	183
Podsumowanie	185
Pytania i odpowiedzi.....	186
Warsztaty.....	186
Quiz.....	186
Ćwiczenie.....	186
Rozdział 13. Instalacja projektów Visual Basic .NET	187
Asemlacje.....	188
Użycie asemlacji	189
Instalowanie asemlacji	190
Asemlacje lokalne	190
Asemlacje WWW	192
Podsumowanie	195
Pytania i odpowiedzi.....	196
Warsztaty.....	197
Quiz.....	197
Ćwiczenie.....	197

Rozdział 14. Praca z interfejsami obiektów	199
Interfejsy i OOP	199
Wielokrotne dziedziczenie	200
Polimorfizm interfejsów	201
Tworzenie interfejsu Contract	202
Programowanie interfejsów	204
Implementacja interfejsów	205
Metody	205
Właściwości	206
Zdarzenia	207
Użycie interfejsu jako wywołania zwrotnego	208
Podsumowanie	210
Pytania i odpowiedzi	210
Warsztaty	211
Quiz	211
Ćwiczenie	211
Rozdział 15. Obsługa zdarzeń, komunikatów i powiadomień	213
Powtórka wiadomości o zdarzeniach i delegatach	214
Obsługa prostych zdarzeń	214
Obsługa zdarzeń dynamicznych za pośrednictwem delegatów	215
Użycie zdarzeń i operacji asynchronicznych	216
Tworzenie i użycie własnych zdarzeń i delegatów	219
Tworzenie własnych zdarzeń z wykorzystaniem delegatów .NET Framework	219
Deklarowanie i użycie własnych delegatów	219
Klasy delegatów	221
Użycie zdarzeń do powiadamiania	221
Podsumowanie	226
Pytania i odpowiedzi	226
Warsztaty	226
Quiz	227
Ćwiczenie	227
Rozdział 16. Przechwytywanie błędów	229
Strukturalna obsługa wyjątków	230
Używanie poleceń Try i Catch	230
Zagnieżdżona obsługa błędów	232
Użycie bloku Finally	233
Generowanie wyjątków	235
Tworzenie i użycie własnych klas wyjątków	238
Podsumowanie	239
Pytania i odpowiedzi	239
Warsztaty	240
Quiz	240
Ćwiczenia	240
Rozdział 17. Projektowanie i używanie obiektów zdalnych	241
Filozofia obiektów zdalnych	241
Obiekty zdalne	242
Obiekty proxy	243
Aktywacja obiektu	244
Cykl życia zdalnego obiektu	244
Dynamiczne publikowanie obiektów zdalnych	245
Wybór pomiędzy kopią a referencją	245
Użycie kanałów w procesie komunikacji	246

Budowa klienta i serwera połączonych za pośrednictwem TCP.....	247
Aplikacja serwera dla obiektów zdalnych	247
Budowanie klienta wykorzystującego zdalny obiekt.....	250
Podsumowanie	253
Pytania i odpowiedzi	253
Warsztaty.....	254
Quiz.....	254
Ćwiczenia.....	254
Rozdział 18. Projektowanie aplikacji zarządzającej projektem	255
Spotkanie z klientem	255
Domeny.....	256
Dekompozycja modelu domen	257
Projektowanie zgodne z wymaganiami.....	258
Tworzenie projektu interakcji	258
Tworzenie klas programowych z klas konceptualnych	259
Podsumowanie	264
Pytania i odpowiedzi.....	264
Warsztaty.....	264
Quiz.....	265
Ćwiczenia.....	265
Rozdział 19. Tworzenie aplikacji zarządzającej projektem	267
Tworzenie modułów klas	267
Tworzenie interfejsu.....	268
Tworzenie klas	269
Klasa Employee	270
Klasy potomne	274
Pozostałe klasy.....	277
Podsumowanie	283
Pytania i odpowiedzi	283
Warsztaty.....	283
Quiz.....	284
Ćwiczenia.....	284
Rozdział 20. Dodawanie cech funkcjonalnych do aplikacji	285
Projektowanie klas wizualnych.....	286
Tworzenie klas wizualnych.....	287
Komponowanie formularza	287
Przeglądanie kodu generowanego.....	290
Tworzenie formularza	291
Rozmieszczanie elementów formularza pracownika	292
Przegląd kodu formularza pracownika	294
Rozmieszczanie elementów formularza produktu	298
Przeglądanie kodu formularza produktu	299
Rozmieszczanie elementów formularza projektu	301
Przeglądanie kodu formularza projektu	302
Rozmieszczanie elementów formularza elementu projektu	305
Przeglądanie kodu formularza elementu projektu	305
Podsumowanie	306
Pytania i odpowiedzi	307
Warsztaty.....	307
Quiz.....	307
Ćwiczenia.....	307

Rozdział 21. Rozszerzanie funkcjonalności aplikacji	309
Dane utrwalone	309
Wprowadzenie do XML-a.....	310
Zasady poprawnego formatowania dokumentu XML	311
Dodatkowe narzędzia.....	312
Document Object Model i XPath	313
DOM	313
Odpytywanie DOM.....	313
Model obiektów DOM.....	314
Kodowanie modułów klas.....	315
Metoda Save().....	315
Metoda Read()	320
Metoda Delete()	322
Klasa Resource.....	322
Przeglądanie danych.....	323
Klasy Product, Project i ProjectItem	324
Podsumowanie	325
Pytania i odpowiedzi	325
Warsztaty.....	325
Quiz.....	326
Ćwiczenia.....	326
Dodatek A Odpowiedzi na pytania quizu	327
Rozdział 1., „Programowanie zorientowane obiektowo — budowa aplikacji sterowanej zadaniami”	327
Quiz.....	327
Ćwiczenia.....	327
Rozdział 2., „Terminologia wykorzystywana w OOP”	330
Quiz.....	330
Ćwiczenia.....	331
Rozdział 3., „Zawieranie danych i cech w obiekcie”	331
Quiz	331
Ćwiczenia.....	331
Rozdział 4., „Tworzenie nowych obiektów na bazie obiektów już istniejących”	332
Quiz.....	332
Ćwiczenia.....	332
Rozdział 5., „Nadawanie obiektom cech polimorficznych”	334
Quiz.....	334
Ćwiczenia.....	334
Rozdział 6., „Budowa obiektów złożonych poprzez łączenie obiektów”	336
Quiz.....	336
Ćwiczenia.....	336
Rozdział 7., „Poznajemy środowisko programistyczne Visual Basic .NET”	337
Quiz.....	337
Ćwiczenia.....	337
Rozdział 8., „Typy danych w Visual Basic .NET”	338
Quiz.....	338
Ćwiczenia.....	338
Rozdział 9., „Organizowanie klas w grupy”	339
Quiz.....	339
Ćwiczenia.....	339
Rozdział 10., „Tworzenie interfejsu użytkownika”	339
Quiz.....	339
Ćwiczenia.....	340

Rozdział 11., „Tworzenie i używanie komponentów”	340
Quiz	340
Ćwiczenia	341
Rozdział 12., „Budowa aplikacji WWW”	342
Quiz	342
Ćwiczenia	342
Rozdział 13., „Instalowanie projektów .NET Visual Basic”	343
Quiz	343
Ćwiczenia	343
Rozdział 14., „Praca z interfejsami obiektów”	344
Quiz	344
Ćwiczenia	344
Rozdział 15., „Obsługa zdarzeń, komunikatów i powiadomień”	344
Quiz	344
Ćwiczenia	345
Rozdział 16., „Przechwytywanie błędów”	346
Quiz	346
Ćwiczenia	346
Rozdział 17., „Projektowanie i używanie obiektów zdalnych”	347
Quiz	347
Ćwiczenia	347
Rozdział 18., „Projektowanie aplikacji zarządzającej projektem”	348
Quiz	348
Ćwiczenia	349
Rozdział 19., „Tworzenie aplikacji zarządzającej projektem”	350
Quiz	350
Ćwiczenia	350
Rozdział 20., „Dodawanie cech funkcjonalnych do aplikacji”	351
Quiz	351
Ćwiczenia	351
Rozdział 21., „Rozszerzanie funkcjonalności aplikacji”	352
Quiz	352
Ćwiczenia	352
Dodatek B Pies... z klasą	355
Tworzenie klasy bazowej	356
Wykorzystanie klasy bazowej	357
Rozwijając „pieską logikę”	358
Podsumowanie	360
Skorowidz	361

Rozdział 14.

Praca z interfejsami obiektów

W poprzednich wersjach Visual Basic'a jedynie interfejsy dawały możliwość zastosowania polimorfizmu. Visual Basic .NET może realizować polimorfizm poprzez interfejsy lub mechanizm dziedziczenia. Użycie interfejsów czy dziedziczenia jest kwestią wyboru pomiędzy realizacją podstawowych i specyficznych funkcji. Zwykle dziedziczenie używane jest w przypadku, gdy podstawowa funkcjonalność realizowana jest w klasach bazowych, natomiast klasy potomne jedynie ją rozszerzają. Interfejsy zaś są optymalnym rozwiązaniem w przypadku, gdy potrzebna jest podobna funkcjonalność w różnych implementacjach kilku klas mających ze sobą niewiele wspólnego.

W rozdziale tym przyjrzymy się bliżej roli interfejsów w Visual Basic .NET i sposobom ich najważniejszego wykorzystania w aplikacjach .NET. Ponieważ w poprzednich rozdziałach omówione zostały podstawowe zagadnienia związane z budowaniem interfejsów, teraz naszą uwagę skupimy na sposobach ich najlepszego wykorzystania. Omówimy więc sposoby:

- ◆ definiowania interfejsów,
- ◆ dziedziczenia pomiędzy interfejsami,
- ◆ implementacji interfejsów wewnątrz definicji klas,
- ◆ rozpoznawania sytuacji, w której wykorzystanie interfejsu jest najlepszym rozwiązaniem,
- ◆ użycia interfejsów do połączeń zwrotnych.

Interfejsy i OOP

Interfejsy, podobnie jak klasy, definiują zbiór właściwości, metod i zdarzeń. Różnica pomiędzy nimi polega na tym, że interfejsy same w sobie nie definiują implementacji, natomiast to klasy, które je wykorzystują, muszą zapewnić ich implementację.

Podstawową zaletą interfejsów jest to, że umożliwiają budowę systemu z opisanych przez interfejsy programowych komponentów, których właściwa implementacja może być zmieniana bez zmiany istniejącego kodu programu korzystającego z tych interfejsów. W rzeczywistości zmianie ulec mogą całe klasy i tak długo, jak implementują one interfejsy, mogą być nadal bez przeszkód używane przez resztę aplikacji.

Wielokrotne dziedziczenie

Podobnie jak ma to miejsce w przypadku klas, interfejsy mogą dziedziczyć po sobie i tworzyć kompletny zorientowany obiektowo projekt. W przypadku interfejsów możliwe jest dziedziczenie po więcej niż jednym obiekcie nadrzędnym, co nie jest możliwe w odniesieniu do klas, a jest to czasem przydatne w projektowaniu zorientowanym obiektowo.

Na przykład *combo box* składa się z pola tekstowego (*test box*) i listy wartości (*list box*). Podczas implementacji *combo box* za pomocą klas niezbędne jest użycie omówionej w rozdziale 6. kompozycji klas, ponieważ niemożliwe jest dziedziczenie klasy typu *combo box* jednocześnie po klasie pól tekstowych i klasie list wartości. Sposób użycia interfejsu *combo box* pokazano na listingu 14.1.

Listing 14.1. Przykładowy interfejs kontrolki *Combo Box*

```
Interface IUIControl
    Sub Paint()
End Interface

Interface ITextBox
    Inherits IUIControl
    Sub SetText(ByVal value As String)
End Interface

Interface IListBox
    Inherits IUIControl

    Sub SetItems(ByVal items() As String)
End Interface

Interface IComboBox
    Inherits ITextBox, IListBox
End Interface
```

Interfejs potomny dziedziczy po interfejsie bazowym wszystkie jego pola. Dlatego też zarówno interfejs *ITextBox*, jak i *IListBox* posiadają podprogram *Paint()*. Interfejs *IComboBox* posiada natomiast podprogramy *SetText()*, *SetItems()* i *Paint()*. Pomimo że podprogram *Paint()* przeniesiony jest z obu interfejsów bazowych, interfejs *IComboBox* ma tylko jedną jego implementację.

Polimorfizm interfejsów

Interfejsy Visual Basic .NET zapewniają kolejny sposób realizacji polimorfizmu. Polimorfizm jest w rzeczywistości nieodłączną cechą interfejsów, ponieważ z założenia nie posiadają one implementacji. Dlatego też każda klasa implementująca interfejs posiada odmienną jego implementację, co stanowi istotę polimorfizmu.

Przykładem polimorfizmu jest definiowanie kilku klas, które implementują ten sam interfejs. Używając interfejsów zamiast klas, zauważamy, w jaki sposób interfejsy obsługują polimorfizm. Listing 14.2 pokazuje użycie interfejsów w serii obiektów będących wielokątami.

Listing 14.2. *Polimorfizm interfejsów*

```
Public Interface Shape
    Function IsPtInShape(ByVal Pt As System.Drawing.Point) As Boolean
End Interface

Public Class Rectangle
    Implements Shape

    Private rect As Drawing.Rectangle

    Public Function PtInRect(ByVal Pt As System.Drawing.Point) As Boolean _
        Implements Shape.IsPtInShape
        PtInRect = Pt.Equals(rect)
    End Function
End Class

Public Class Circle
    Implements Shape

    Private rgn As Drawing.Region

    Public Function PtInCircle(ByVal Pt As System.Drawing.Point) As Boolean _
        Implements Shape.IsPtInShape
        PtInCircle = Pt.Equals(rgn)
    End Function
End Class

Sub CheckPoint(ByVal ShapeObject As Shape, ByVal Pt As System.Drawing.Point)
    If (ShapeObject.IsPtInShape(Pt)) Then
        Beep()
    End If
End Sub

Sub Test()
    Dim Pt As System.Drawing.Point
    Dim RectObject As Rectangle
    Dim CircleObject As Circle

    CheckPoint(RectObject, Pt)
    CheckPoint(CircleObject, Pt)
End Sub
```

analiza

Zauważmy, w jaki sposób podprogram `CheckPoint()` pobiera `Shape` jako parametr, a następnie używa metody interfejsu. W dalszej kolejności wywoływana jest metoda implementująca funkcjonalność interfejsu. Metoda ta jest różna dla każdego typu obiektu. Możemy nazwać tę metodę dowolnie i implementować w sposób, jaki najbardziej nam w określonej sytuacji odpowiada, pod warunkiem zachowania zgodności listy parametrów. W ten sposób klasa może implementować interfejs, zachowując jednocześnie niezmienną nazwę metody wewnątrz klasy. Możliwe jest również zastąpienie dowolnego obiektu implementującego interfejs `Shape` jako parametru podprogramu `CheckPoint()`.

Tworzenie interfejsu Contract

Interfejs reprezentuje „umowę” z klasą. Klasa, używając słowa kluczowego `Implements`, „przystaje na umowę” mówiąc o tym, że będzie implementować interfejs. Po wyrażeniu przez klasę chęci implementacji interfejsu jest ona zobowiązana do implementacji całego interfejsu. Klasa nie może zaimplementować części interfejsu, wypełniając jednocześnie zaakceptowany wcześniej kontrakt. Dlatego też, jeśli klasa implementuje interfejs, użytkownicy klasy pewni są tego, że mogą używać wszystkich metod zdefiniowanych w tym interfejsie, ponieważ z całą pewnością zostaną one zaimplementowane.

Należy pamiętać, że klasa może wypełnić umowę, dostarczając w szczególnym przypadku takiej implementacji, która nie wykonuje żadnej czynności. Jedynym warunkiem jest zgodność w stosunku do zapisów zdefiniowanych w interfejsie dotyczących parametrów i (w przypadku funkcji) zwracanych wartości. Wszystko inne zależy od klasy implementującej interfejs. W rzeczywistości zaletą użycia interfejsów jest to, że klasa decyduje o zastosowaniu określonej implementacji w konkretnym przypadku. Przykładowa klasa przedstawionej na listingu 14.3 implementuje trzy różne interfejsy.

Listing 14.3. *Implementacja wielu interfejsów przez jedną klasę oraz użycie obiektów klasy jako implementacji każdego z interfejsów*

```
Public Interface IEmployee
    Function GetSalary() As Decimal
End Interface

Public Interface IParent
    Function HasChildren() As Boolean
End Interface

Public Interface IFriend
    Function NumberOfFriends() As Integer
End Interface

Public Class Person
    Implements IEmployee
    Implements IParent
    Implements IFriend
```

```
Private m_dSalary As Decimal
Private m_bChildren As Boolean
Private m_nFriends As Integer

Public Function Salary() As Decimal Implements IEmployee.GetSalary
    Salary = m_dSalary
End Function

Public Function HasChildren() As Boolean Implements IParent.HasChildren
    HasChildren = m_bChildren
End Function

Public Function HowManyFriends() _
As Integer Implements IFriend.NumberOfFriends
    HowManyFriends = m_nFriends
End Function
End Class

Module PersonUser
Sub CheckSalary(ByVal Employee As IEmployee)
    ' W tym miejscu należy umieścić kod programu..
    ' Tylko GetSalary() może być używana dla tego obiektu
End Sub

Sub CheckFamily(ByVal Parent As IParent)
    ' W tym miejscu należy umieścić kod programu..
    ' Tylko HasChildren() może być używana dla tego obiektu
End Sub

Sub CheckFriends(ByVal MyFriend As IFriend)
    ' W tym miejscu należy umieścić kod programu..
    ' Tylko NumberOfFriends() może być używana dla tego obiektu
End Sub

Sub TestPerson()
    Dim Person As Person = New Person()

    CheckSalary(Person)
    CheckFamily(Person)
    CheckFriends(Person)
End Sub
End Module
```

Dla kompletności definicji klasy `Person` niezbędne jest zaimplementowanie wszystkich metod zdefiniowanych w interfejsach `IEmployee`, `IParent` i `IFriend`. Klasa `Person` definiować może również własne cechy charakterystyczne, jednakże te metody, właściwości i zdarzenia nie będą dostępne, gdy klasa będzie używana poprzez te interfejsy.

W każdym z trzech interfejsów pokazanych w podprogramie `TestPerson()` możliwe jest użycie klasy `Person`. Jest to szczególnie użyteczne podczas projektowania aplikacji posiadającej kilka niezależnych lub luźno powiązanych klas, które używane będą zamiennie. Stosując typ interfejsu jako parametr, użytkownik widzi jedynie metody, właściwości i zdarzenia zdefiniowane w interfejsie. Reszta implementacji wewnątrz klasy nie jest dla użytkownika interfejsu widoczna.

Programowanie interfejsów

Poznaliśmy już kilka przykładów zastosowania interfejsów w programowaniu zorientowanym obiektowo. Programowanie z wykorzystaniem interfejsów może jednak nieco różnić się od programowania z użyciem klas.

Pierwszym spostrzeżeniem poczynionym podczas projektowania interfejsów, które mają być implementowane później, jest to, że ich metody nie mają określonych zakresów dostępności. Wszystkie metody są jawne i dostępne do użytku jako publiczne. Na przykład wewnątrz klasy zdefiniować możemy metodę publiczną, chronioną lub prywatną. Interfejsy nie mają tej cechy. Każda metoda zdefiniowana w interfejsie dostępna jest jako publiczna, czyli tak, jakby została zadeklarowana jako `Public`. Nawet jeśli klasa implementująca implementuje metody jako `Private` dla użytkownika korzystającego z interfejsu, są one dostępne tak, jakby były zadeklarowane jako `Public`.

Posiadanie wszystkich metod interfejsu zadeklarowanych jako publiczne na sens, jeśli zdefiniowanie interfejsu było gruntownie przemyślane, gdyż interfejsy definiowane są w celu opisanego wspólnych możliwych do użycia cech czy funkcjonalności. Czy więc może znaleźć się w interfejsie coś, co zostało zadeklarowane jako prywatne? Działanie takie nie ma żadnego logicznego wytłumaczenia z wyjątkiem tego, że mogłoby to wymuszać na klasie implementację metody prywatnej, która może być użyta tylko w obrębie danej klasy.

Kolejnym zagadnieniem, które na pierwszy rzut oka wydaje się być problemem (choć w rzeczywistości okazuje się to nieprawdą), jest fakt, że interfejs nie definiuje pól danych. Zadaniem interfejsu jest definiowanie jedynie publicznych interfejsów metod, zdarzeń i właściwości. Ponieważ, jak pamiętamy, interfejsy nie mają implementacji, nie ma sensu definiowanie pól danych, gdyż dostęp do danych powinien być realizowany poprzez ich metody i właściwości.

Ostatnim problemem, który omówimy, jest nieco frustrująca sytuacja, gdy dwa interfejsy mają zaprojektowaną tę samą metodę. Jeśli dwa interfejsy są interfejsami bazowymi dla nowego, trzeciego interfejsu, użytkownik zmuszony jest do określenia, którą z metod chce wywołać. Listing 14.4 pokazuje taki właśnie przypadek.

Listing 14.4. *Interfejs z kolizją nazw używających CType() do wskazania właściwego sposobu użycia metod*

```
Public Interface IArray
    Function Count() As Integer
End Interface

Public Interface ICounter
    Function Count() As Integer
End Interface

Public Interface ICountArray
    Inherits IArray
    Inherits ICounter
End Interface
```

```
Module CountArrayTest
Sub Test(ByVal x As ICountArray)
    Dim nCount = CType(x, ICounter).Count()
    Dim nArrayCount = CType(x, IArray).Count()
    'Pozostały kod programu..
End Sub
End Module
```

analiza W kodzie z listingu 14.4 funkcja `CType()` konwertuje wyrażenie precyzujące pierwszy argument (w tym przypadku `x`) do typu używanego w drugim parametrze. Dzięki temu podczas wykonywania kodu wybierany jest właściwy interfejs i używana jest właściwa metoda `Count`.

Nieco wcześniej wspominaliśmy, że jeśli ta sama metoda zawarta jest wewnątrz interfejsu i w jego interfejsie bazowym, to jest ona implementowana tylko raz. Dlatego też nie występuje wtedy problem pokazany na listingu 14.4.

Implementacja interfejsów

Ponieważ interfejs nie posiada implementacji, musi być przed użyciem zdefiniowany tak, aby dostarczał odpowiednich funkcji, metod, zdarzeń i właściwości. Interfejs może być implementowany zarówno przez klasy, jak i definicje struktur zawierających słowo kluczowe `Implements`.

Jak już wspominaliśmy, klasa może implementować jeden lub wiele interfejsów. Przedstawiony poniżej kod pokazuje sposób, w jaki struktura implementuje interfejs podobnie, jak robi to klasa.

```
Structure MyStructure
Implements IMyInterface

Public Function InterfaceFunction() _
    As Integer Implements IMyInterface.MyFunction
End Function
End Structure
```

Jak już wspominaliśmy wcześniej, interfejs może deklarować metody (podprogramy i funkcje), zdarzenia i właściwości. Pamiętajmy jednak, że nie może deklarować pól danych.

Metody

W celu implementacji metod interfejsów poprzez klasy lub struktury, metody te deklarowane są wewnątrz klas lub struktur z wykorzystaniem słowa kluczowego `Implements`. Jedynym wymaganie implementacji metody interfejsu jest to, aby metoda klasy lub struktury posiadała te same parametry i zwracała wartości zadeklarowane w metodzie interfejsu.

Tak długo jak zgodne są listy parametrów i wartości zwracanych danych, możliwe jest również implementowanie wielu interfejsów przez pojedynczą metodę klasy lub struktury.

Na przykład, zamieszczony poniżej kod pokazuje, jak w jednej funkcji zaimplementować dwie metody `Count()` zdefiniowane w listingu 14.4:

```
Public Function Count() As Integer Implements IArray.Count(), ICounter.Count()  
    Count() = m_nCount  
End Function
```

Poprzez implementowanie wielu metod interfejsów za pomocą tej samej metody klasy lub struktury możliwe jest powiązanie metod wielu interfejsów z tą samą funkcjonalnością bez konieczności pisania dodatkowego kodu.

Właściwości

Implementacja właściwości interfejsu podobna jest do implementacji metod. Realizowane jest to za pomocą słowa kluczowego `Implements` przypisującego właściwości klas lub struktur do właściwości interfejsów. Właściwości klas implementujące właściwości interfejsu muszą charakteryzować się tym samym typem (zapis/odczyt). Listing 14.5 ilustruje implementację właściwości interfejsu.

Listing 14.5. *Implementacja właściwości interfejsu*

```
Public Interface ICompany  
    Property Name() As String  
End Interface  
  
Public Class Business  
    Implements ICompany  
  
    Dim m_sCompanyName As String  
  
    Public Property CompanyName() As String Implements ICompany.Name  
        Get  
            CompanyName = m_sCompanyName  
        End Get  
        Set(ByVal Value As String)  
            m_sCompanyName = Value  
        End Set  
    End Property  
  
End Class
```

Jeśli właściwość zadeklarowana jest w interfejsie jako `ReadOnly` lub `WriteOnly`, to implementująca go klasa lub struktura musi implementować te właściwości z takimi samymi atrybutami.

Zdarzenia

Podobnie jak w przypadku metod i właściwości, zdarzenia implementowane są poprzez klasy lub struktury za pomocą słowa kluczowego `Implements`. Jak wspominaliśmy już w rozdziale 2., zdarzenia w definicjach klas nie są powiązane z kodem. Skutkiem użycia słowa kluczowego `Implements` jest literalne połączenie zdarzenia klasy lub struktury ze zdarzeniem interfejsu. Listing 14.6 pokazuje, jak implementować i używać zdarzenia za pośrednictwem interfejsów.

Listing 14.6. *Implementacja i użycie zdarzeń interfejsu*

```
Public Interface ICompany
    Property Name() As String
    Event NameChanged()
End Interface

Public Class Business
    Implements ICompany

    Dim m_sCompanyName As String

    Public Property CompanyName() As String Implements ICompany.Name
        Get
            CompanyName = m_sCompanyName
        End Get
        Set(ByVal Value As String)
            m_sCompanyName = Value

            'Sygnał o zdarzeniu - nazwa została zmieniona
            RaiseEvent NameChanged()
        End Set
    End Property

    Public Event NameChanged() Implements ICompany.NameChanged

End Class

Module CompanyTest
    Sub OnNameChanged()
        Beep()
    End Sub

    Sub Test(ByVal Company As ICompany)
        'Dodanie uchwytu interfejsu zdarzenia NameChanged
        AddHandler Company.NameChanged, AddressOf OnNameChanged

        'Ten fragment kodu powoduje wygenerowanie zdarzenia
        Company.Name = "My Company"
    End Sub
End Module
```

Użycie interfejsu jako wywołania zwrotnego

Używane poprawnie interfejsy mają wiele zalet. Mamy możliwość tworzenia interfejsów wolnych od problemów występujących podczas zmiany kodu bazowego. Prowadzi to również do większej elastyczności implementacji.

Innym przykładem użycia interfejsów jest zdefiniowane wywołanie zwrotnego (*callback*). Na przykład, jeśli kilka typów implementuje ten sam interfejs, może być on wykorzystany jako metoda wywołania zwrotnego w sytuacji, w której pojedyncze wywołanie zwrotne jest niewystarczające. Użycie interfejsu pozwala na wywołanie zwrotne zawierające kilka metod, właściwości lub zdarzeń. Listing 14.7 przedstawia sposób użycia interfejsu zdefiniowanego jako wywołanie zwrotne. Ilekroć interfejs ten jest implementowany, obiekt implementujący go wie, jaki interfejs powinien wywołać zwrotnie. W odróżnieniu od pojedynczej funkcji lub podprogramu wywołanie zwrotne jest kompletnym interfejsem.

Listing 14.7. Deklaracja wywołania zwrotnego sterowania interfejsem użytkownika

```
Public Interface UIControl
    Sub Paint()
    Sub ControlPressed()
    Function IsPtInControl(ByVal Pt As Drawing.Point) As Boolean
    Property ControlRect() As Drawing.Rectangle
    Event ControlHit()
End Interface
```

Implementacja interfejsu `IUIControl` poprzez różne klasy dostarcza wspólnego interfejsu używanego podczas pracy z kontrolkami. Listing 14.8 pokazuje prostą implementację kontrolki przycisku (`PushButton`) i przycisku opcji (`RadioButton`).

Listing 14.8. Implementacja interfejsu `IUIControl` poprzez klasy `PushButton` i `RadioButton`

```
Public Class PushButton
    Implements UIControl

    Dim m_rcRect As Drawing.Rectangle

    'Implementacja interfejsu
    '.....
    Public Sub Paint() Implements UIControl.Paint
        ' Rysowanie kontrolki
    End Sub

    Public Function IsPtInControl(ByVal Pt As Drawing.Point) _
        As Boolean Implements UIControl.IsPtInControl
        IsPtInControl = Pt.Equals(ControlRect())
    End Function

    Public Property ControlRect() As Drawing.Rectangle _
        Implements UIControl.ControlRect
```

```
    Get
        ControlRect = m_rcRect
    End Get
    Set(ByVal Value As Drawing.Rectangle)
        m_rcRect = Value
    End Set
End Property

Public Sub ControlPressed() Implements IUIControl.ControlPressed
    RaiseEvent ControlHit()
End Sub

Public Event ControlHit() Implements IUIControl.ControlHit

'Pozostała część kodu klasy
'.....

End Class

Public Class RadioButton
    Implements IUIControl

    Dim m_rcRect As Drawing.Rectangle

    'Implementacja interfejsu
    '.....
    Public Sub Paint() Implements IUIControl.Paint
        ' Rysowanie kontrolki
    End Sub

    Public Function IsPtInControl(ByVal Pt As Drawing.Point) _
        As Boolean Implements IUIControl.IsPtInControl
        IsPtInControl = Pt.Equals(ControlRect())
    End Function

    Public Property ControlRect() As Drawing.Rectangle _
        Implements IUIControl.ControlRect
        Get
            ControlRect = m_rcRect
        End Get
        Set(ByVal Value As Drawing.Rectangle)
            m_rcRect = Value
        End Set
End Property

    Public Sub ControlPressed() Implements IUIControl.ControlPressed
        RaiseEvent ControlHit()
    End Sub

    Public Event ControlHit() Implements IUIControl.ControlHit

    'Pozostała część kodu klasy
    '.....

End Class
```

Implementacja interfejsu `IUIControl` zależy od kontroltek i każda z kontroltek może mieć swoją własną implementację. Używanie kontroltek jest więc teraz ułatwione, ponieważ użytkownicy, chcąc wywołać zwrócić kontrolkę, kontaktują się jedynie z interfejsem. Ilustruje to listing 14.9.

Listing 14.9. *Użycie interfejsu `IUIControl` do wywołania zwrócić do klasy w przypadku wystąpienia zdarzenia*

```
Public Class MyForm
    Inherits System.Windows.Forms.Form

    Dim UIControls(2) As Object

    Public Sub OnMouseClicked(ByVal sender As Object, _
        ByVal e As EventArgs) Handles MyBase.Click
        Dim i As Integer

        For i = 0 To 2
            If CType(UIControls(i), IUIControl).IsPtInControl(MousePosition()) Then
                CType(UIControls(i), IUIControl).ControlPressed()
                CType(UIControls(i), IUIControl).Paint()
            End If
        Next
    End Sub
End Class
```

Oczywiście możliwa jest również współpraca interfejsów z asynchronicznymi wywołaniami zwrócić służącymi do komunikacji sieciowej, dostępu do plików itp.

Podsumowanie

Poznaliśmy właśnie sposoby użycia interfejsów w Visual Basic .NET, jako części programowania zorientowanego obiektowo. Zaletą interfejsów, w porównaniu z klasami, jest możliwość dziedziczenia z kilku interfejsów bazowych jednocześnie. Interfejsy są z natury polimorficzne, ponieważ każda implementująca je klasa lub struktura tworzy własną, oryginalną ich implementację. Klasy i struktury mogą implementować więcej niż jeden interfejs, co umożliwia ich wielokrotne użycie w aplikacji.

Pytania i odpowiedzi

Pytanie: Po co tworzyć interfejsy? Czy nie można tego wszystkiego zrobić za pomocą klas, skoro one również wykorzystują metodę dziedziczenia i polimorfizmu?

Odpowiedź: To prawda, interfejsy nie są dziś wykorzystywane tak często jak w poprzednich wersjach Visual Basica, gdy dziedziczenie i polimorfizm nie były możliwe na poziomie klas. Jednakże mają one nadal przewagę nad klasami, ponieważ

umożliwiają definiowanie ogólnych interfejsów implementowanych przez niezależne obiekty, umożliwiają wielokrotne dziedziczenie i sprawiają, że kod jest niewrażliwy na ewentualne przyszłe modyfikacje.

Pytanie: Czy interfejsy w .NET uległy dużym zmianom? Czy możemy nadal używać ich poza aplikacją lub eksportować ?

Odpowiedź: Interfejsy w Visual Basic .NET są generalnie podobne do tych z poprzednich wersji Visual Basic. .NET Framework ukrywa wiele szczegółów implementacyjnych, ale w rzeczywistości bazuje on nadal na identyfikatorach GUID, a dane, przechodząc przez interfejs, nadal są przekształcane.

Warsztaty

W ramach zajęć warsztatowych proponować będziemy zagadki i ćwiczenia, które pomogą uporządkować i przećwiczyć materiał poznany w kończącym się właśnie rozdziale książki. Odpowiedzi przedstawiono w dodatku A na końcu książki.

Quiz

1. Jakie słowo kluczowe deklaruje interfejs?
2. Jakie słowo kluczowe łączy metody, właściwości i zdarzenia z interfejsem?
3. Czy klasa może implementować część interfejsu?
4. Ile interfejsów może implementować klasa?
5. Jaki inny typ poza klasą może implementować interfejs?

Ćwiczenie

Stwórz interfejs `Error` mający metodę o nazwie `DisplayError()`, wyświetlającą komunikat o błędzie. Następnie zadeklaruj klasę `MyClass` implementującą interfejs `Error` i wyświetlającą komunikat o błędach podany w parametrze metody `DisplayError()`.