

Wykorzystaj
potęgę
technologii
Windows!

Jacek Matulewski

VISUAL BASIC

.NET w praktyce

Błyskawiczne tworzenie aplikacji



Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Redaktor prowadzący: Ewelina Burska

Projekt okładki: Magdalena Stasik

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

Wydawnictwo HELION

ul. Kościuszki 1c, 44-100 GLIWICE

tel. 32 231 22 19, 32 230 98 63

e-mail: helion@helion.pl

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie?vbnepr>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Dodatkowe materiały do książki są dostępne pod adresem:

<ftp://ftp.helion.pl/przyklady/vbnepr.zip>

ISBN: 978-83-246-4898-6

Copyright © Helion 2012

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Wstęp	11
Część I Projektowanie aplikacji Windows	13
Rozdział 1. Środowisko Visual Studio	15
Projektowanie interfejsu aplikacji	15
Tworzenie projektu	16
Dokowanie palety komponentów Toolbox	18
Tworzenie interfejsu za pomocą komponentów Windows Forms	19
Zapisywanie i wczytywanie projektu	20
Ukryta prawda	21
Analiza kodu pierwszej aplikacji	22
Metody zdarzeniowe	26
Metoda uruchamiana w przypadku wystąpienia zdarzenia kontrolki	26
Testowanie metody zdarzeniowej	27
Przypisywanie istniejącej metody do zdarzeń komponentów	29
Edycja metody zdarzeniowej	30
Modyfikowanie własności komponentów	30
Wywoływanie metody zdarzeniowej z poziomu kodu	31
Reakcja aplikacji na naciskanie klawiszy	32
Platforma aplikacji (application framework)	32
Rozdział 2. Debugowanie kodu	35
Skąd biorą się błędy i jak ich unikać?	35
Kontrolowane uruchamianie aplikacji	36
Program z błędem — pole do popisu dla debuggera	36
Śledzenie wykonywania programu krok po kroku (klawisze F10 i F11)	37
Run to Cursor (klawisze Ctrl+F10)	38
Breakpoint (klawisz F9)	39
Okna Locals i Watch	40
Stan wyjątkowy	42
Przechwytywanie wyjątków w konstrukcji try..catch	43
Zgłaszanie wyjątków	45

Rozdział 3. Język Visual Basic	47
Platforma .NET	47
Trivia	49
Podstawowe typy danych	49
Deklaracja i zmiana wartości zmiennej	49
Typy liczbowe oraz znakowy	50
Opcje Strict, Infer i Explicit	53
Priorytety operatorów	56
Weryfikacja typów	58
Łańcuchy	59
Weryfikowanie zgodności łańcucha ze wzorcem	62
Typ wyliczeniowy	63
Leniwe inicjowanie zmiennych	65
Funkcje i metody	65
Przeciążanie metod	66
Domyślne wartości argumentów metod — argumenty opcjonalne	67
Wartości zwracane przez metody	68
Zwracanie wartości przez argument metody (ByRef)	68
Delegacje i zdarzenia	70
Wyrażenia lambda	71
Typy wartościowe i referencyjne	72
Nullable	74
Pudełkowanie	75
Sterowanie przepływem	76
Instrukcja warunkowa If.Else	76
Instrukcja wyboru Select	77
Pętle	77
Wyjątki	79
Dyrektywy preprocesora	82
Kompilacja warunkowa — ostrzeżenia	82
Definiowanie stałych preprocesora	83
Bloki	83
Atrybuty	84
Kolekcje	84
„Zwykłe” tablice	85
Pętla foreach	88
Sortowanie	89
Kolekcja List	90
Kolekcja SortedList i inne słowniki	91
Kolejka i stos	92
Rozdział 4. Projektowanie zorientowane obiektowo	95
Przykład struktury (Ułamek)	96
Przygotowanie projektu	96
Konstruktor i współdzielone obiekty składowe	97
Pierwsze testy	98
Konwersje na łańcuch (metoda ToString) i na typ double	99
Metoda upraszczająca ułamek	99
Własności	100
Operatory arytmetyczne	101
Operatory porównania oraz metody Equals i GetHashCode	103
Operatory konwersji	104
Implementacja interfejsu (na przykładzie IComparable)	105

Definiowanie typów parametrycznych	107
Definiowanie typów ogólnych	108
Określanie warunków, jakie mają spełniać parametry	109
Implementacja interfejsów przez typ ogólny	110
Definiowanie aliasów	112
Typy ogólne z wieloma parametrami	113
Rozszerzenia	114
Słowo kluczowe With	116
Typy anonimowe	117
Rozdział 5. Przegląd komponentów biblioteki Windows Forms	119
Notatnik.NET	119
Projektowanie interfejsu aplikacji i menu główne	120
Okna dialogowe i pliki tekstowe	126
Edycja i korzystanie ze schowka	134
Drukowanie	135
Elektroniczna kukułka	145
Ekran powitalny (splash screen)	145
Przygotowanie ikony w obszarze powiadamiania	148
Odtwarzanie pliku dźwiękowego	151
Ustawienia aplikacji	153
Tworzenie ustawień w trakcie projektowania aplikacji	153
Odczytywanie ustawień z poziomu kodu	154
Zapisywanie ustawień z poziomu kodu	155
Dywan graficzny	156
Zdarzenie Paint formy	156
Kolorowy wzór	157
Buforowanie	158
Zapisywanie obrazu dywanu do pliku	159
Rysowanie zbiorów Mandelbrota i Julii	160
Trochę teorii	160
Implementacja	163
Lista uruchomionych procesów	166
Rozdział 6. Przeciągnij i upuść	169
Podstawy	169
Interfejs przykładowej aplikacji	170
Inicjacja procesu przeciągania	171
Akceptacja upuszczenia elementu	173
Reakcja na upuszczenie elementu	174
Czynności wykonywane po zakończeniu procesu przenoszenia i upuszczania	175
Przenoszenie elementów między różnymi aplikacjami	176
Zagadnienia zaawansowane	176
Opóźnione inicjowanie procesu przenoszenia	176
Przenoszenie wielu elementów	179
Przenoszenie plików	181
Rozdział 7. Przezroczyste okna o dowolnym kształcie	183
Konfiguracja formy	183
Wczytywanie obrazu	184
Przezroczystość i łagodne znikanie okna	186
Zamykanie klawiszem Esc	187
Przenoszenie formy za dowolny punkt	188
Menu kontekstowe	189

Rozdział 8. Projektowanie kontroltek	193
Komponent FileListBox	194
Implementacja podstawowych funkcjonalności	194
Rozbudowa komponentu o możliwość zmiany katalogu	203
Właściwości	205
Zdarzenia — interakcja z komponentem	210
Odświeżanie komponentu i automatyczne śledzenie zmian w prezentowanym katalogu	216
Kompilacja komponentu do postaci biblioteki DLL	219
Prosty przykład wykorzystania komponentu FileListBox: przeglądanie plików tekstowych	224
Kolorowy pasek postępu	228
Tworzenie projektu	228
Rysowanie obramowania kontrolki	229
Poła i własności	230
Rysowanie paska postępu	233
Metody	234
Zdarzenia	235
Rozdział 9. Zarządzane biblioteki DLL i mechanizm rozpoznawania typów	239
Tworzenie zarządzanej biblioteki DLL	240
Projekt biblioteki DLL	240
Dodawanie referencji do biblioteki systemowej platformy .NET	241
Wyświetlanie informacji o systemie i platformie .NET	242
Zmiana przestrzeni nazw biblioteki	243
Statyczne ładowanie bibliotek DLL	243
Dołączanie do projektu bibliotek DLL użytkownika	243
Dynamiczne ładowanie zarządzanych bibliotek DLL i dynamiczne rozpoznawanie typów	245
Dynamiczne ładowanie zarządzanej biblioteki .dll	245
Analiza zawartości biblioteki załadowanej dynamicznie	246
Weryfikacja obecności w bibliotece DLL klasy o znanej nazwie	247
Lista metod w klasie z biblioteki DLL	248
Weryfikacja obecności konkretnej metody w klasie z biblioteki DLL	249
Lista argumentów wybranej metody	250
Uruchamianie metody współdzielonej z klasy wczytanej z biblioteki DLL	252
Uruchamianie metody na rzecz instancji obiektu. Przekazywanie parametrów i odczytywanie zwracanej wartości	254
Późne wiązanie na typie Object	255
Rozdział 10. Aplikacje konsolowe i informacje o systemie	257
Klasa Console	257
Projekt aplikacji konsolowej	257
Drukowanie napisów w konsoli	259
Czekanie na akceptację użytkownika	259
Odczytywanie danych z klawiatury	260
Komunikat „okienkowy” w aplikacji konsolowej	261
Informacje o środowisku aplikacji	262
Podstawowe informacje o systemie i profilu użytkownika	263
Katalogi specjalne zdefiniowane w bieżącym profilu użytkownika	264
Odczytywanie zmiennych środowiskowych	264
Lista dysków logicznych	265

Rozdział 11. Wątki, równoległa pętla Parallel.For oraz programowanie asynchroniczne (Async/Await)	267
Monte Carlo	268
Obliczenia bez użycia dodatkowych wątków	269
Przeniesienie obliczeń do osobnego wątku	270
Uspianie wątku	272
Przerywanie działania wątku (Abort)	273
Wstrzymywanie i wznowianie działania wątku	274
Wątki działające w tle	275
Zmiana priorytetu wątku	276
Użycie wielu wątków i problemy z generatorem liczb pseudolosowych	276
Czekanie na ukończenie pracy wątku (Join)	280
Sekcje krytyczne (lock)	282
Przesyłanie danych do wątku	283
Pula wątków	285
Jeszcze raz o komunikacji między wątkami	288
Korzystanie z muteksów w celu zapobiegania uruchamianiu wielu instancji aplikacji	290
Klasa Parallel z biblioteki TPL	290
Równoległa pętla For	291
Przerywanie pętli	293
Programowanie asynchroniczne (Async/Await)	294
Rozdział 12. Podstawy ADO.NET i SQL Server	297
Bardzo krótki wstęp do SQL	298
Select	298
Insert	299
Delete	299
Aplikacje „bazodanowe”	299
Projekt aplikacji z bazą danych	299
Konfiguracja komponentu DataSet	302
Server Explorer	304
Podgląd danych udostępnianych przez komponent DataSet	304
Prezentacja danych w siatce DataGridView	305
Edycja danych	307
Projektowanie formularzy prezentujących pojedyncze rekordy	309
Sortowanie	312
Filtrowanie	312
Odczytywanie z poziomu kodu wartości przechowywanych w komórkach	313
Część II LINQ	315
Rozdział 13. Wprowadzenie do zapytań LINQ na przykładzie kolekcji (LINQ to Objects)	317
Pobieranie danych (filtrowanie i sortowanie)	319
Najprostsza prezentacja pobranych danych	320
Analiza pobranych danych	320
Wybór elementu	320
Weryfikowanie danych	321
Prezentacja w grupach	321
Łączenie zbiorów danych	322
Łączenie danych z różnych źródeł w zapytaniu LINQ — operator join	322
Możliwość modyfikacji danych źródła	323

Rozdział 14. LINQ to DataSet	325
Konfiguracja kontrolki DataSet	326
LINQ to DataSet, czyli tam i z powrotem	328
Rozszerzenie IEnumerable klasy DataTable	332
Obliczenia wykonywane na danych z tabeli	332
Dowolność sortowania i filtrowania pobieranych danych	332
Rozdział 15. LINQ to SQL	335
Klasa encji	336
Pobieranie danych	337
Aktualizacja danych w bazie	338
Modyfikacje istniejących rekordów	339
Dodawanie i usuwanie rekordów	340
Inne operacje	341
Wizualne projektowanie klasy encji	342
O/R Designer	342
Współpraca z kontrolkami tworzącymi interfejs aplikacji	346
Kreator źródła danych i automatyczne tworzenie interfejsu użytkownika	347
Łączenie danych z dwóch tabel — operator join	349
Relacje (Associations)	351
Korzystanie z procedur składowanych	353
Pobieranie danych za pomocą procedur składowanych	353
Modyfikowanie danych za pomocą procedur składowanych	355
Wykonywanie dowolnych poleceń SQL	355
Rozdział 16. Kilka sposobów na odczytywanie i zapisywanie danych w plikach XML	357
Podstawy języka XML	357
Deklaracja	358
Elementy	358
Atrybuty	359
Komentarze	359
Klasy XmlTextReader i XmlTextWriter	359
Zapis do pliku XML	359
Odczyt danych z pliku XML	361
Analiza i odczyt pliku XML o nieznannej strukturze	363
Serializacja obiektów do pliku XML	365
Serializacja obiektu do pliku XML	366
Deserializacja obiektu z pliku XML	367
XML i ADO.NET	368
Wczytywanie danych z pliku XML do komponentu DataSet	369
Zapisywanie zmian do pliku XML za pośrednictwem DataSet	370
LINQ to XML	371
Tworzenie pliku XML za pomocą klas XDocument i XElement	371
Pobieranie wartości z elementów o znanej pozycji w drzewie	373
Przenoszenie danych z kolekcji do pliku XML	375
Przenoszenie danych z bazy danych (komponentu DataSet) do pliku XML	377
Zapytania LINQ	378
Modyfikacja pliku XML	379

Część III Technologie Windows	381
Rozdział 17. Rejestr systemu Windows	383
Korzystanie z rejestru	383
Odczytywanie danych z rejestru	383
Zapisywanie oraz odczytywanie położenia i rozmiaru formy w prywatnym kluczu aplikacji	386
Usuwanie klucza z rejestru	388
Przeglądarka skojarzeń plików	389
Informacja o typach plików przechowywana w rejestrze	389
Przygotowanie interfejsu	391
Odczytywanie listy rozszerzeń	392
Pobieranie opisu, polecenia głównego i domyślnego edytora dla podanego typu plików	394
Rozdział 18. Mechanizm PInvoke	399
Funkcja bez argumentów	399
Problemy z argumentami	401
Zwracanie wartości przez argumenty	403
Zwracanie tablicy znaków w funkcjach WinAPI	406
Rozdział 19. Komunikaty Windows	407
Wysyłanie komunikatów Windows	407
Identyfikacja aplikacji	408
Wysyłanie komunikatu do okna o znanym uchwycie	409
Komunikaty jako sposób porozumiewania się z systemem	410
Odbieranie komunikatów Windows	411
Monitor komunikatów	411
Reakcja na wybrany komunikat	412
Skorowidz	415

Rozdział 2.

Debugowanie kodu

Skąd biorą się błędy i jak ich unikać?

Za błędy w kodzie jest (niemal) wyłącznie odpowiedzialny programista. Ale to oczywiste stwierdzenie niewiele nam pomaga w ich unikaniu. Zresztą błędów w ogóle nie można unikać. Ich powstawanie jest nierozzerwalnie związane z samym procesem tworzenia kodu. Co więcej, liczba błędów w kodzie zależy proporcjonalnie od liczby linii kodu, a współczynnik proporcjonalności jest powiązany ze stopniem doświadczenia i niewyspania programisty. Prosty wniosek jest taki, że pisany przez nas kod powinien mieć jak najmniejszą liczbę linii. Do tego celu trzeba dążyć za pomocą wszelkich możliwych sposobów i sztuczek. Przede wszystkim należy unikać powtarzania kodu. Lepiej przygotować jedną metodę, którą możemy porządnie przetestować, niż powielać jej fragmenty w kilku miejscach, co uniemożliwia ich kontrolę w przypadku jakichkolwiek modyfikacji. Zatem lepiej wywołać porządnie napisaną metodę, zamiast korzystać z „kopiuj i wklej”. Poza tym, jeżeli możemy użyć gotowego przetestowanego kodu (np. kontrolki), to warto z tego skorzystać. W tworzeniu kodu może nas też wyręczyć środowisko programistyczne. W przypadku aplikacji Windows Forms dzieje się tak zresztą już w momencie, gdy bawiąc się myszą, budujemy interfejs aplikacji.

Z pozornie bezużytecznego faktu, że błędów w kodzie nie da się unikać, co dobitnie wyraża prawo Murphy’ego: „Nie ma aplikacji bez błędów”, można wyciągnąć jeszcze jeden wniosek: otóż, skoro występowanie błędów jest niemal pewne, trzeba się na nie przygotować. Nie można zakładać, że błędy będą zdarzać się na tyle rzadko, iż prawdopodobieństwo ich wystąpienia będzie tak małe, że nie będą dla użytkownika uciążliwe. Inne prawo Murphy’ego mówi bowiem: „Jeżeli błąd może wystąpić, wystąpi na pewno, i to w takim momencie, gdy użytkownikowi dopieczy to najbardziej” (np. gdy kończąc pracę nad dokumentem, będzie chciał zapisać go do pliku). Jak wspomniałem, jest zatem konieczne przygotowanie programu na możliwość wystąpienia błędów. Korzystajmy wobec tego z obsługi wyjątków w każdej budzącej wątpliwość sytuacji, nawet tam, gdzie — jak się nam wydaje — usunęliśmy już wszystkie usterki. Przemyślana reakcja na wyjątki może pozwolić na zminimalizowanie skutków wystąpienia błędu, czasem nawet do tego stopnia, że użytkownik nie musi się dowiedzieć, iż coś złego się wydarzyło.

Teoria Murphy’ego wypowiada się także na temat najmniej lubianej czynności programistów, a więc na temat testowania i tropienia błędów logicznych z kodu. Jak już wiemy, nie ma aplikacji bezbłędnych, liczbę błędów można jedynie redukować. Należy jednak zdawać sobie sprawę, że tzw. znalezienie ostatniego błędu to idea regulatywna debugowania. Należy w nią wierzyć, ale z góry wiadomo, że jest nieosiągalna. Warto też uświadomić sobie zaskakującą, ale w świetle teorii Murphy’ego oczywistą prawdę: poprawiając błędy w kodzie, tworzymy nowe.

Kontrolowane uruchamianie aplikacji

Na szczęście, w procesie usuwania błędów (debugowania) kodu Visual Basic dla platformy .NET mamy potężnego sprzymierzeńca. Jest nim środowisko programistyczne Visual Studio z wbudowanym debuggerem, wskazującym linie kodu, które nie podobają się kompilatorowi, pozwalającym na kontrolę uruchamianego kodu, a nawet na śledzenie jego wykonywania linia po linii. Przyjrzyjmy się bliżej kilku jego możliwościom.

Program z błędem — pole do opisu dla debuggera

Zacznijmy od przygotowania aplikacji, której metody zawierają błąd powodujący wystąpienie wyjątku. W tym celu:

1. Utwórz nowy projekt typu *Windows Forms Application* o nazwie *Debugowanie*.
2. W widoku projektowania umieść na formie przycisk *Button*.
3. Kliknij dwukrotnie ów przycisk, aby utworzyć domyślną metodę zdarzeniową (w tym przypadku do zdarzenia *Button1.Click*). Zostaniesz automatycznie przeniesiony do edytora kodu.
4. Zanim uzupełnisz zawartość metody zdarzeniowej, utwórz definicję metody *Kwadrat* (umieść ją np. bezpośrednio przed utworzonym automatycznie „szkieletem” metody *Button1_Click*, listing 2.1).

Listing 2.1. *Metoda, w której ukryliśmy perfidny błąd*

```
Public Class Form1

    Private Function Kwadrat(argument As Integer) As Integer
        Dim wartosc As Integer
        wartosc = argument * argument
        Return wartosc
    End Function

    Private Sub Button1_Click(sender As System.Object, e As System.EventArgs)
        ↪Handles Button1.Click

    End Sub
End Class
```

5. Teraz przejdź do metody `Button1_Click` i wpisz do niej polecenia wyróżnione w listingu 2.2.

Listing 2.2. *Z pozoru wszystko jest w porządku...*

```
Private Sub Button1_Click(sender As System.Object, e As System.EventArgs) Handles  
    ↳Button1.Click  
        Dim x As Integer = 1234  
        Dim y As Integer = Kwadrat(x)  
        y = Kwadrat(y)  
        Dim sy As String = y.ToString()  
        MessageBox.Show("Wynik: " & sy)  
    End Sub
```

Oczywiście, obie metody można by „skompresować” do jednej lub dwóch linii, ale właśnie taka forma ułatwi naukę debugowania.

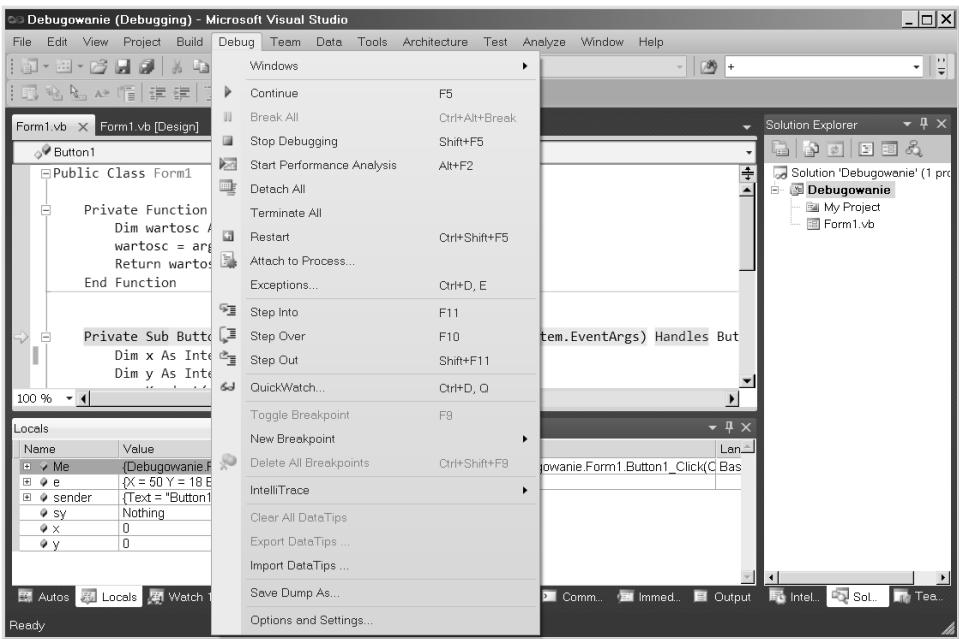
Uruchomimy najpierw aplikację (klawisz *F5*), żeby przekonać się, że nie działa prawidłowo. Po kliknięciu przycisku zamiast komunikatu z wynikiem zobaczymy komunikat o wyjątku z informacją, że nastąpiło przekroczenie zakresu zmiennej (*Arithmetic operation resulted in an overflow*). Miejsce wystąpienia wyjątku wskaże nam środowisko Visual Studio — program zatrzyma się na mnożeniu w metodzie `Kwadrat`. Nie jest jednak oczywiste, które wywołanie tej metody (a wywoływana jest dwa razy) prowadzi do błędu. Mamy zatem do czynienia z błędem logicznym ukrytym gdzieś w naszym kodzie. I to właśnie jego tropienie będzie motywem przewodnim większej części tego rozdziału.

Śledzenie wykonywania programu krok po kroku (klawisze *F10* i *F11*)

Naciśnijmy klawisz *F10* lub *F11* (wszystkie klawisze skrótów wykorzystywane podczas debugowania zebrane zostały w tabeli 2.1). Jeżeli włączona jest wspomniana w poprzednim rozdziale platforma aplikacji (ang. *application framework*), to aplikacja uruchomi się normalnie. Dopiero gdy klikniemy przycisk na formie, jej działanie zatrzyma się i zobaczymy zaznaczoną na żółto sygnaturę metody zdarzeniowej związanej z tym zdarzeniem (rysunek 2.1). Inaczej jest, gdy platforma aplikacji jest wyłączona, a my zdefiniowaliśmy własną metodę `Main`. Wówczas po naciśnięciu *F10* lub *F11* na żółto zaznaczona zostanie jej sygnatura, a środowisko będzie czekało na dalsze polecenia. Każde naciśnięcie klawisza *F10* powoduje wówczas wykonanie jednej linii kodu (tej zaznaczonej na żółto), bez względu na to, czy jest to inicjacja zmiennej, czy wywołanie metody. Czynność taka nazywa się *Step Over* (menu *Debug*), czyli „krok nad”. Nazwa bierze się z tego, że jeżeli w wykonywanej linii znajduje się wywołanie metody, jest ona wykonywana w całości. Natomiast *F11* powoduje wykonanie „kroku w głąb” (ang. *step into*), co w przypadku wywołania metody oznacza, że zostaniemy przeniesieni do pierwszej linii jej definicji i tam będziemy kontynuować śledzenie działania aplikacji. W przypadku gdy ta metoda zawiera wywołanie kolejnej metody, klawisze *F10* i *F11* pozwolą zdecydować, czy chcemy ją wykonać w całości, czy przeanalizować linia po linii. Jeżeli zorientujemy się, że zeszliliśmy zbyt głęboko — możemy nacisnąć *Shift+F11*, aby wykonać pozostałą część metody i ją opuścić (ang. *step out* — wyjście z).

Tabela 2.1. Związane z debugowaniem klawisze skrótów środowiska Visual Basic

Funkcja	Klawisz skrótu
Uruchomienie z debugowaniem	F5
Uruchomienie bez debugowania	Ctrl+F5
Uruchomienie i zatrzymanie w linii, w której jest kursor	Ctrl+F10
Krok do następnej linii kodu (<i>Step over</i>)	F10
Krok z wejściem w głąb metody (<i>Step into</i>)	F11
Krok z wyjściem z metody (<i>Step out</i>)	Shift+F11
Ustawienie <i>breakpointu</i> (funkcja edytora)	F9
Zakończenie debugowania (zakończenie działania aplikacji)	Shift+F5

**Rysunek 2.1.** Na rysunku rozwinięte jest menu Debug, zawierające instrukcje sterujące procesem kontrolowanego uruchamiania aplikacji

Run to Cursor (klawisze Ctrl+F10)

W przypadku gdy platforma aplikacji jest wyłączona i mamy zdefiniowaną metodę Main, korzystanie z klawiszy F10 i F11 może nie być zbyt wygodne. Dla przykładu, jeżeli interesuje nas tylko kod metody zdarzeniowej Button1_Click, to przechodzenie krok po kroku przez zawartość metody Main jest całkowicie zbędne. Wówczas wygodniej jest posłużyć się kombinacją klawiszy Ctrl+F10. Podobnie jak klawisz F5, uruchamia ona aplikację, ale zatrzymuje ją w momencie, gdy wykonana ma być instrukcja z linii, w której znajduje się kursor edytora. Ta kombinacja klawiszy działa także wtedy, gdy aplikacja jest już uruchomiona w trybie debugowania. Po zatrzymaniu

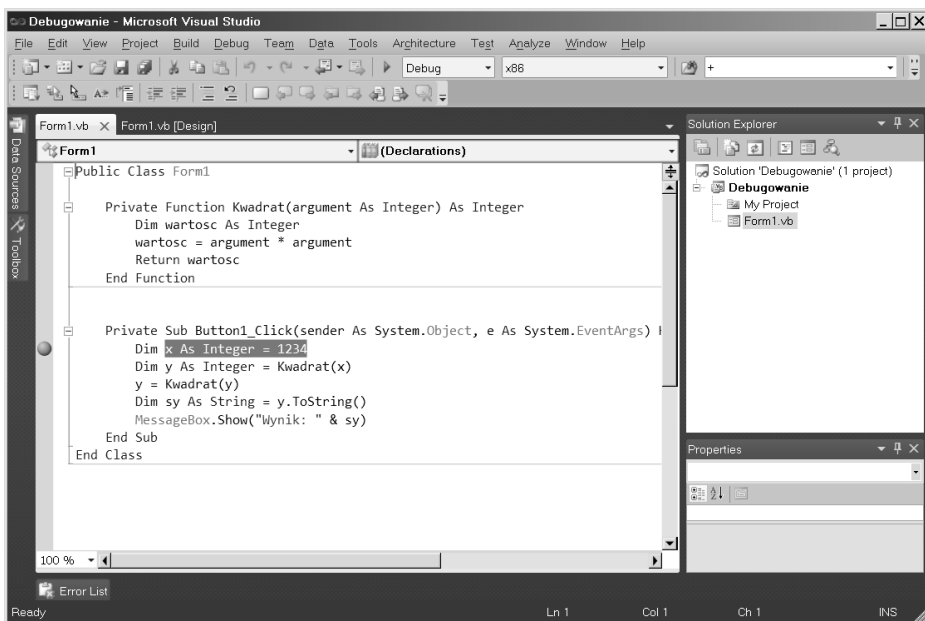
działania aplikacji znowu możemy korzystać z klawiszy *F10* i *F11* do śledzenia działania metod krok po kroku.



Aby natychmiast przerwać debugowanie programu i powrócić do normalnego trybu edycji Visual Studio, należy nacisnąć klawisze *Shift+F5*.

Breakpoint (klawisz F9)

Gdy przewidujemy, że będziemy wielokrotnie kontrolować wykonywanie pewnych poleceń, np. z metody `Button1_Click`, możemy w jej pierwszej linii ustawić tzw. *breakpoint*. W tym celu przechodzimy do wybranej linii w edytorze kodu i naciskamy klawisz *F9*. Linia zostanie zaznaczona bordowym kolorem oraz czerwoną kropką na lewym marginesie (rysunek 2.2). Po uruchomieniu programu w trybie debugowania jego działanie zostanie zatrzymane, gdy wątek dotrze do tej linii. Możemy wówczas obejrzeć wartości zmiennych (o tym za chwilę), przejść do śledzenia kodu (klawisze *F10* i *F11*) lub nacisnąć klawisz *F5*, aby wznowić jego działanie w normalnym trybie debugowania. Gdy jednak wątek znowu dotrze do linii z ustawionym *breakpointem*, działanie programu jeszcze raz zostanie wstrzymane. Aby anulować *breakpoint*, należy ustawić kursor w odpowiedniej linii i jeszcze raz nacisnąć *F9* lub kliknąć widoczną na lewym marginesie czerwoną kropkę.

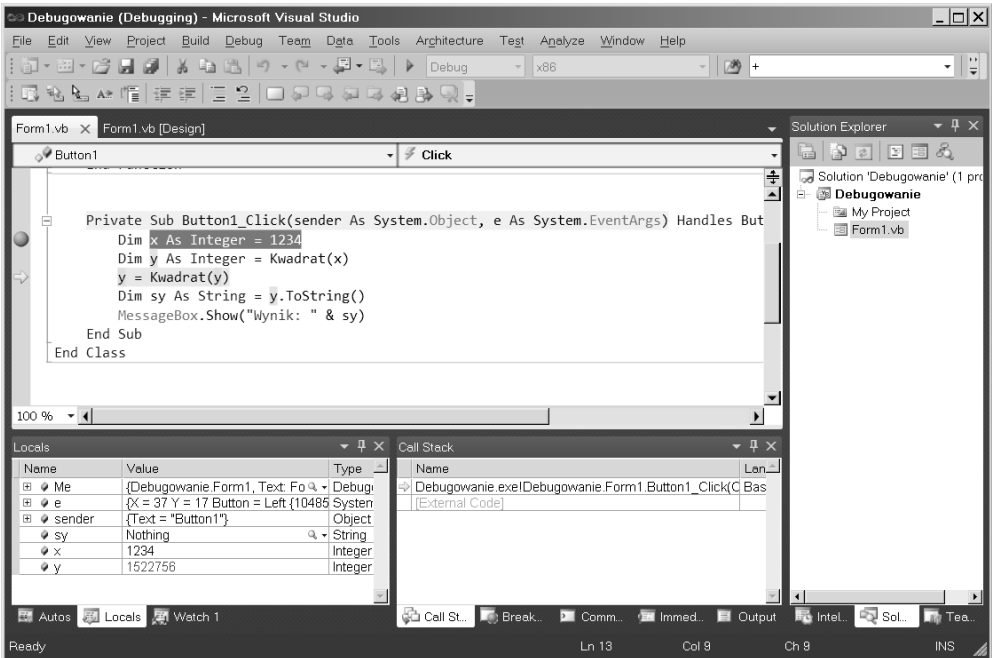


Rysunek 2.2. Wyróżnienie linii kodu, w której ustawiony jest breakpoint

Breakpoint bywa szczególnie pożyteczny przy śledzeniu wykonywania pętli. Jeżeli ustawimy go w interesującej nas linii wewnątrz pętli, każda jej iteracja zostanie przerwana, co pozwala np. na przyjrzenie się wartościom zmiennych.

Okna Locals i Watch

Możliwość obserwacji wartości zmiennych używanych w programie to bardzo ważne narzędzie debuggerów. Dzięki niemu możemy w każdej chwili sprawdzić, czy wartości zmiennych są zgodne z naszymi oczekiwaniami, a to pozwala ocenić, czy program działa prawidłowo. Wykonywanie aplikacji linia po linii z równoczesnym wpatrywaniem się w wartości zmiennych jest w praktyce najczęściej wykorzystywanym sposobem na odszukanie owego „ostatniego błędu”. W Visual Studio służy do tego okno *Locals* (rysunek 2.3), które zawiera listę wszystkich pól zadeklarowanych wewnątrz obiektów i zmiennych lokalnych zadeklarowanych w aktualnie wykonywanej metodzie wraz z ich wartościami. Jeżeli zmienne są obiektami, możemy zobaczyć także ich pola składowe. Poza tym mamy do dyspozycji okno *Watch*, do którego możemy dodać nie tylko poszczególne pola i zmienne, ale również poprawne wyrażenia języka Visual Basic, np. x^*x .

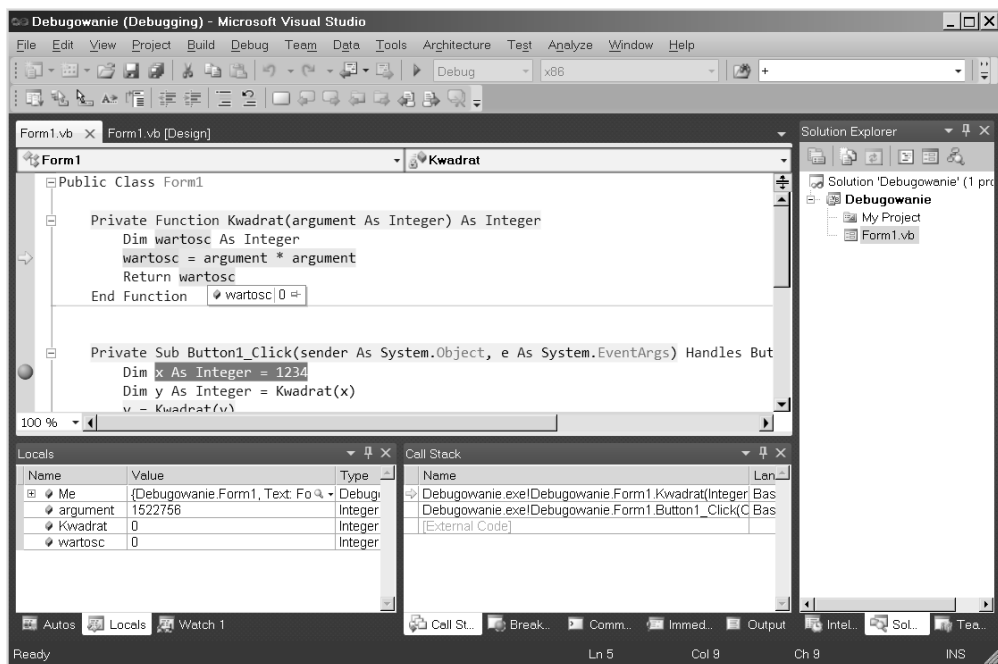


Rysunek 2.3. Podglądanie wartości zmiennych w edytorze



Uwaga

Wartości zmiennych można również zobaczyć, jeżeli w trybie debugowania w edytorze przytrzymamy przez chwilę kursor myszy nad zmienną widoczną w kodzie. Po chwili pojawi się okienko podpowiedzi zawierające wartość wskazanej w ten sposób zmiennej (rysunek 2.4). Należy jednak pamiętać, że pokazywana wartość dotyczy aktualnie wykonywanej linii, a nie linii wskazanej kursorem. W Visual Studio 2010 taką podejrzaną w oknie edytora wartość można przypiąć (ikoną pinezki), przez co podczas pracy w trybie debugowania pozostaje stale widoczna na ekranie (rysunek 2.4). Możliwość podglądania wartości zmiennych w oknie edytora dotyczy także obiektów — zobaczymy wówczas wartości wszystkich dostępnych pól i własności.



Rysunek 2.4. U dołu okna VS widoczne są dwa podokna: z lewej okno *Locals*, z prawej — *Call Stack*

Przejdźmy w edytorze kodu (zakładka *Form1.vb*) do linii w metodzie *Button1_Click*, w której zdefiniowana jest zmienna *y*. Zaznaczmy tę zmienną i prawym przyciskiem myszy rozwińmy menu kontekstowe. Z niego wybierzmy polecenie *Add Watch* (dostępne tylko, gdy uruchomione jest debugowanie). Zmienna ta zostanie dodana do listy w oknie *Watch*, w której zobaczymy jej nazwę, wartość i typ. Wartość jest aktualizowana przy każdym naciśnięciu klawisza *F10* lub *F11*, co pozwala na śledzenie jej zmian w trakcie wykonywania metody. Możemy się przekonać, wykonując kolejne polecenia metody *Button1_Click* ponownymi naciśnięciami klawisza *F10*, że po pierwszym uruchomieniu metody *Kwadrat* zmienna *y* ma wartość równą 1522756. Jest zatem poprawna. Niestety, przy drugim uruchomieniu metody *Kwadrat* pojawi się wyjątek. Aby sprawdzić, co jest przyczyną błędu, wchodzimy do jej „środką”, używając klawisza *F11*. W metodzie *Kwadrat* nie ma zmiennej *y*, a jej wartość przejmuje zmienna *argument*. Dodajmy ją zatem do listy obserwowanych zmiennych, wpisując jej nazwę do pierwszej kolumny w podoknie *Watch*. Dodajmy także wyrażenie *argument*argument*. W trakcie drugiego przebiegu metody *Kwadrat* przy tym wyrażeniu pojawi się komunikat *Constant expression not representable in type „Integer”*, czyli „wartość nie może być przedstawiona w zmiennej typu »Integer«”.

Informacja ta wyjaśnia przyczynę błędu — podczas mnożenia został przekroczony, i to znacznie, zakres możliwych wartości zmiennej *Integer* ($1\ 522\ 756 \cdot 1\ 522\ 756 = 2\ 318\ 785\ 835\ 536 > 2\ 147\ 483\ 647$). Innymi słowy, 32 bity, jakie oferuje typ *Integer*, nie wystarczają do zapisania wyniku. Aby poprawić ów błąd, przynajmniej w pewnym zakresie wartości argumentów, należałoby użyć 64-bitowego typu *Long*. Ponadto, ponieważ kwadrat liczby całkowitej nie może być ujemny, warto pozbyć się także znaku

i użyć typu `ULong`. Kod pozbawiony błędu widoczny jest na listingu 2.3. Na razie tych zmian jednak nie wprowadzamy — korzystając z obecności błędu, chciałbym zaprezentować mechanizm przechwytywania wyjątków.

Listing 2.3. Poprawiony kod, którego na razie nie wstawiamy do projektu

```
Private Function Kwadrat(argument As Integer) As Long
    Dim wartosc As ULong
    wartosc = CULng(argument) * CULng(argument)
    Return wartosc
End Function

Private Sub Button1_Click(sender As System.Object, e As System.EventArgs) Handles
↳Button1.Click
    Dim x As Integer = 1234
    Dim y As ULong = Kwadrat(x)
    y = Kwadrat(y)
    Dim sy As String = y.ToString()
    MessageBox.Show("Wynik: " & sy)
End Sub
```



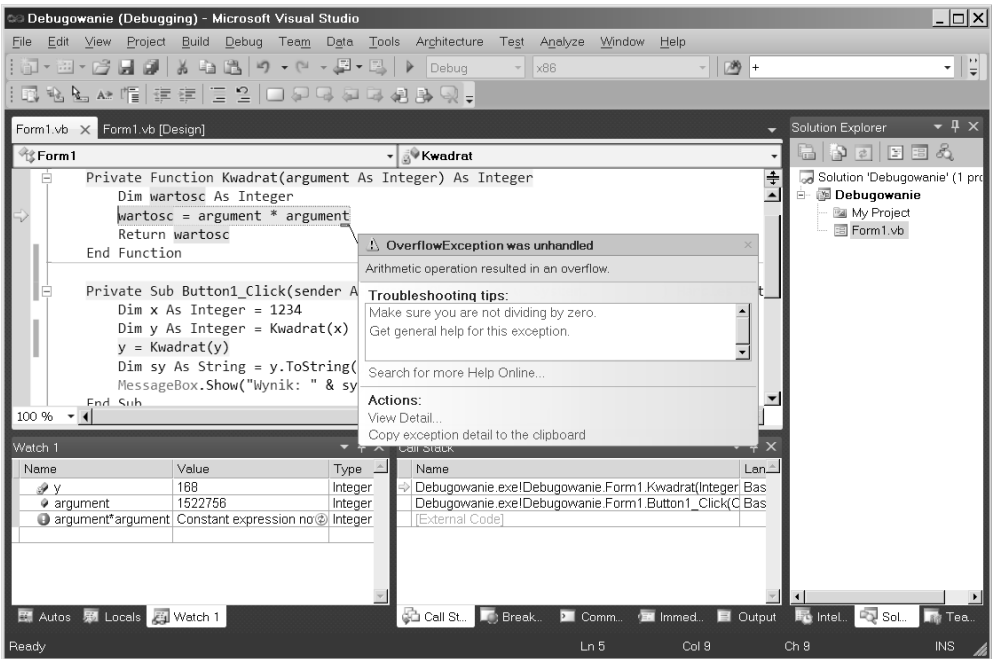
Uwaga

Na rysunku 2.4 z prawej strony na dole widoczne jest okno *Call Stack* (z ang. stos wywołań). Wymienione są w nim wszystkie metody, począwszy od metody `Button1_Click`, a skończywszy na metodzie, której polecenie jest obecnie wykonywane. W bardziej rozbudowanych programach pomaga to w zorientowaniu się, co się aktualnie dzieje w aplikacji, szczególnie po jej zatrzymaniu w wyniku działania *breakpointu*.

Stan wyjątkowy

Jeżeli nie wprowadziliśmy zmian z listingu 2.3 do projektu, w razie uruchomieniu projektu z debugowaniem (klawisz *F5*) po kliknięciu przycisku `Button1` pojawi się komunikat debuggera informujący o wyjątku (rysunek 2.5). Przyjrzyjmy się temu komunikatowi. Przede wszystkim na pasku tytułu widoczne jest ostrzeżenie *OverflowException was unhandled*, co oznacza, że wyjątek został zgłoszony przez fragment kodu, który nie był otoczony obsługą wyjątków. Nie znajdował się zatem w sekcji `try` konstrukcji `try...catch` ani w metodzie, która byłaby z tej sekcji wywołana. W efekcie w przypadku uruchomienia aplikacji poza środowiskiem Visual Studio wyjątek ten nie byłby w żaden sposób obsłużony przez aplikację i musiałaby się nim zająć sama platforma .NET, co raczej nie powinno mieć miejsca. W oknie komunikatu debuggera widoczna jest także treść przekazywanego przez wyjątek komunikatu oraz link do dokumentacji klasy wyjątku (w naszym przypadku klasy `OverflowException`). Po kliknięciu *View Detail...* można obejrzeć stan przesyłanego obiektu w widocznym na dole okna odpowiedniku okna *Locals*.

Po wystąpieniu wyjątku środowisko Visual Studio wstrzymuje działanie aplikacji na tej samej zasadzie jak *breakpoint*, tzn. zwykle możemy przywrócić jej działanie, np. za pomocą klawiszy *F5* lub *F10*. Możemy także zupełnie przerwać działanie aplikacji i przejść do poprawiania kodu, naciskając kombinację klawiszy *Shift+F5*.



Rysunek 2.5. Komunikat o wyjątku zgłoszony przez Visual Studio

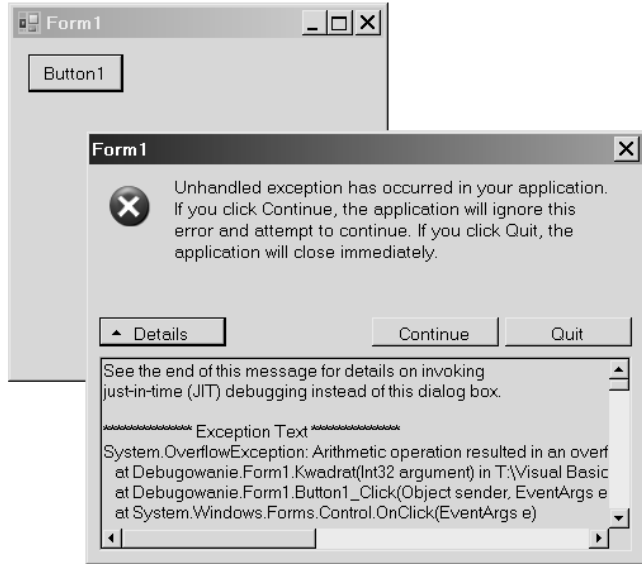
Jeżeli aplikacja zostanie uruchomiona poza debuggerem przez naciśnięcie kombinacji klawiszy *Ctrl+F5* lub samodzielne uruchomienie pliku *.exe* poza środowiskiem Visual Studio, nieobsłużony wyjątek jest przechwytywany przez platformę .NET. O takiej sytuacji użytkownik powiadamiany jest komunikatem widocznym na rysunku 2.6. Proszę zwrócić uwagę na to, że w przeciwieństwie do platformy Win32, w której analogiczny komunikat oznaczałby awaryjne zamknięcie aplikacji, tu istnieje możliwość kontynuowania jej działania (przycisk *Kontynuuj* widoczny na rysunku 2.6). Oczywiście, wyjątek pojawi się znowu, jeżeli klikniemy przycisk *Button1* na formie i uruchomiona zostanie metoda *Kwadrat*, ale aplikacja nie zawiesza się. Warto docenić tę własność platformy .NET — jeżeli nasza aplikacja byłaby edytorem, a błąd pojawiałby się np. podczas drukowania, platforma .NET, nie zamykając całej aplikacji w przypadku wystąpienia błędu, dałaby nam możliwość m.in. zachowania niezapisanego dokumentu na dysku.

Przechwytywanie wyjątków w konstrukcji *try..catch*

Teraz spróbujmy wykryć wystąpienie wyjątku z poziomu aplikacji. W tym celu w metodzie zdarzeniowej przycisku *Button1_Click* otoczmy wywołanie metody *Kwadrat* konstrukcją przechwytywania wyjątków.

1. Naciśnij *Shift+F5*, aby zakończyć działanie debugera.
2. Do metody *Button1_Click* dodaj obsługę wyjątku zgodnie ze wzorem na listingu 2.4.

Rysunek 2.6.
Wyjątek nieobsłużony
w aplikacji jest
przechwytywany przez
środowisko .NET
(aplikacja uruchomiona
jest bez debuggera)



Listing 2.4. Wszystkie dotychczasowe polecenia metody `Button1_Click` umieściliśmy w jednej sekcji `Try`

```
Private Sub Button1_Click(sender As System.Object, e As System.EventArgs) Handles
↳Button1.Click
    Try
        Dim x As Integer = 1234
        Dim y As Integer = Kwadrat(x)
        y = Kwadrat(y)
        Dim sy As String = y.ToString()
        MessageBox.Show("Wynik: " & sy)
    Catch ex As Exception
        MessageBox.Show("Błąd: " & ex.Message,
            "Przechwycony wyjątek w metodzie Button1_Click",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error)
    End Try
End Sub
```

3. Skompiluj i uruchom aplikację w trybie debugowania (klawisz *F5*).

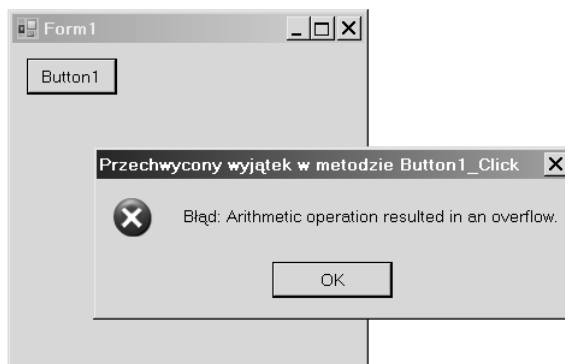
Teraz po uruchomieniu aplikacji i kliknięciu przycisku zamiast komunikatu debuggera lub komunikatu platformy .NET zobaczymy własny komunikat o błędzie (rysunek 2.7). Debugger nie zainterweniuje w przypadku obsługowanego wyjątku.

Wszystkie polecenia metody `Button1_Click` umieściliśmy w jednej sekcji `Try`. To ma sens, gdy każde wystąpienie błędu powoduje, że wykonywanie dalszej części metody pozabawione jest celu. Jeśli natomiast po pojawieniu się błędu chcielibyśmy podjąć jakąś akcję ratunkową i próbować kontynuować działanie metody, to każda budząca wątpliwości instrukcja powinna być otoczona oddzielną konstrukcją obsługi wyjątków.



Dodatkowe informacje na temat wyjątków można znaleźć w rozdziale 3.

Rysunek 2.7.
*Komunikat wyświetlany
użytkownikowi po
przechwyceniu wyjątku*



Zgłaszanie wyjątków

Ostatnią rzeczą, o której chciałbym wspomnieć w kontekście wyjątków, jest ich zgłaszanie. Jeżeli w kodzie wykryjemy błąd, dla przykładu jeżeli sprawdzimy, że argument metody Kwadrat jest zbyt duży, to możemy samodzielnie zgłosić wyjątek. W takim przypadku należy użyć instrukcji `Throw` zgodnie ze wzorem widocznym na listingu 2.5. Kod ten używa dodatkowego pola (zmiennej zdefiniowanej w obrębie klasy) o nazwie `MaxArgumentKwadrat`, której definicja również widoczna jest na poniższym listingu.

Listing 2.5. *Wartość pierwiastka obliczana jest tylko raz, więc kontrola argumentu nie obciąża nadmiernie procesora*

```
Dim MaxArgumentKwadrat As Integer = Math.Sqrt(Integer.MaxValue)

Private Function Kwadrat(argument As Integer) As Integer
    Dim wartosc As Integer
    If argument < MaxArgumentKwadrat Then
        Throw New Exception("Argument metody kwadrat ma zbyt dużą wartość")
    End If
    wartosc = argument * argument
    Return wartosc
End Function.
```

Gdy teraz po uruchomieniu aplikacji klikniemy przycisk `Button1`, zamiast dotychczasowego komunikatu informującego o przepełnieniu zobaczymy nasz własny, bardziej konkretny komunikat.

Skorowidz

A

ADO.NET, 297, 368
aktualizacja danych, 338
aktualizowanie ścieżki, 218
algorytm
 generujący zbiór Julii, 165
 testowania, 162
alias, 112
analiza pliku XML, 363
aplikacja *Patrz* projekt
aplikacje
 konsolowe, 261
 okienkowe, 16
 Windows Forms, 80
argumenty metody, 251
atrybut, 84, 359
 DllImport, 84
 Extension, 114
atrybuty elementów, 365
automatyczne
 mapowanie struktury, 342
 tworzenie klasy encji, 345

B

baza danych, 297
 Access, 304
 Telefony.mdf, 350
biblioteka
 DLL, 84
 Metro, 12
 STL, 107
 System.Drawing.dll, 224
 System.Numerics, 163
 System.Windows.Forms.dll,
 224
 SystemInfo.dll, 253
 TPL, 267

User32.dll, 401
VCL, 171
VCL.NET, 171
Windows Forms, 119, 171
WPF, 12, 119
biblioteki
 .dll, 193, 239
 .NET, 47
 .ocx, 241
bloki kodu, 83
błąd, 278
 kompilacji, 54
 logiczny, 37
 w kodzie pętli, 91
breakpoint, 39
budowanie tabeli, 330
bufor obiektu, 340
buforowanie, 158

C

CAS, Code Access Security, 48
CLR, Common Language
 Runtime, 48
CLS, Common Language
 Specification, 48
CTS, Common Type System, 48

D

dane
 aktualizowanie, 338
 analizowanie, 320
 filtrowanie, 332
 grupowanie, 321
 łączenie zbiorów, 322
 prezentacja, 320
 sortowanie, 332
 weryfikowanie, 321

debugowanie, 36
definicja
 delegacji, 70
 typu
 Okno, 366
 WielkośćOkna, 366
 uchwyty do metody, 70
 właściwości
 CzyKatalogiWidoczne, 207
 Filtr, 207
 ŚcieżkaDoZaznaczonego-
 Elementu, 208
definiowanie
 aliasów, 112
 symboli preprocesora, 83
 tablicy, 85
 typów ogólnych, 108
 własnego zdarzenia, 211
deklaracja pola typu Panel, 24
deklaracja zmiennej, 49
deserializacja z pliku XML, 367
długie linie, 141
dodawanie
 pól do klasy, 188
 kontrolki, 195
 referencji do biblioteki,
 163, 241
 rekordów, 340
dokowanie palety, 18
domyślna wartość argumentu, 67
dostęp do pola, 25
dostęp przez referencję Me, 127
drukowanie, 135
drukowanie w tle, 144
dymek, balloon tip, 150
dynamiczne ładowanie
 bibliotek, 239

dyrektywa

- #Const, 83
- #End Region, 83
- #If, 82
- #Region, 83

dyrektywy preprocesora, 82
dystrybucja biblioteki, 221

E

edycja

- bazy danych, 301
- menu kontekstowego, 190
- pliku XML, 369

edytor

- głównego menu, 125
- kodu, 21
- menu kontekstowego, 148
- relacji, 351
- własności Dock, 121

ekran powitalny, splash

- screen, 145

element listy, 92

elementy, 358

encje, entity, 359

F

fabryka, 65

fazy procesu drag & drop, 175

fazy przenoszenia elementu, 174

FIFO, first in, first out, 92

FILO, first in, last out, 92

filtr okna dialogowego, 132

filtrowanie, 201, 312, 352

filtrowanie wyjątków, 80

filtry, 131

flaga daneZmienione, 307

focus, 25

forma

- kolor, 184
- kształt, 184

formularz, 309, 311

funkcja

- CreateProcess, 403
- FindWindow, 409
- MessageBeep, 399, 400, 401
- MsgBox, 49
- SendMessage, 409
- ShellExecute, 403
- WinAPI, 404
- WinExec, 402

funkcje bez argumentów, 399

G

garbage collector, 62

generator liczb pseudolosowych,
77, 276, 279

GUI, graphical user interface,
15, 119

I

identyfikacja aplikacji, 408

IL, Intermediate Language, 48

implementacja

- interfejsu, 106
- interfejsu IComparable, 110
- listy, 90

import funkcji WinAPI, 408

importowanie przestrzeni
nazw, 83

informacje o

- kontrolce, 222
- systemie, 253, 263
- ścieżkach, 384
- typie aplikacji, 395

inicjowanie zmiennej, 54

instancja klasy, 95

instrukcja

- Continue For, 79
- If..Else, 76
- If..Then, 32
- Select, 77

IntelliSense, 28, 118

interfejs

- aplikacji, 348, 392
- graficzny, 15
- IComparable, 90
- IComparable(Of T), 110
- IConvertible, 99
- IDictionary, 92
- IEnumerable, 332
- IEnumerable(Of), 315
- komponentu, 194
- przeglądarki, 225
- WinExec, 403

izomorficzność relacji, 336

J

jawne deklarowanie

- zmiennych, 12

język funkcyjny F#, 47

JIT, Just-In-Time, 48

K

kanał alpha, 30

kataglogi specjalne, 265

klasa, 95

- AdresyDataSet, 303, 304
 - Application, 127
 - CategoryAttribute, 205
 - ColorProgressBar, 232
 - Console, 257
 - DataContext, 335, 341
 - DataTable, 332
 - DescriptionAttribute, 205
 - encji, entity class, 336
 - Environment, 264
 - EventWaitHandle, 288, 289
 - Exception, 130
 - FileListBox, 196
 - Form1, 22, 177
 - Graphics, 136, 229
 - Lazy, 65
 - Monitor, 288
 - Mutex, 290
 - OverflowException, 42
 - Panel, 19, 25
 - Para, 113
 - Parallel, 267, 290, 292
 - ParameterizedThreadStart, 283
 - Process, 167
 - Program, 264
 - Queue, 91
 - Random, 77
 - Regex, 62
 - Registry, 392
 - RegistryKey, 385
 - SoundPlayer, 152
 - Stack, 92
 - StreamReader, 130
 - String, 60
 - StringBuilder, 61, 406
 - System.Array, 85
 - System.Enum, 63
 - System.Environment, 262
 - System.IO.DriveInfo, 404
 - SystemInfo, 242, 243, 244
 - Thread, 267
 - ThreadPool, 285
 - TrackBar, 19
 - XDocument, 372, 374
 - XmlTextReader, 359
 - XmlTextWriter, 361
 - zagnieżdżona, 246
- klasy, 72
- opakowujące, 50
 - rozszerzające, 116

- klawisz Enter, 259
 - klawisze skrótów
 - VB, 28
 - do debugowania, 38
 - menu, 125
 - klucz, key, 92
 - główny, 383, 393
 - prywatny, 397
 - prywatny aplikacji, 386
 - rejestr, 384
 - użytkownika, 395
 - kod
 - klasy Form1, 22
 - klasy kontrolki, 197
 - niezarządzany, unmanaged code, 240
 - niezrównoległony, 269
 - pośredni MSIL, 48
 - zarządzany, managed code, 48
 - zbioru Mandelbrota, 164
 - kolejka, queue, 91
 - kolekcja, 84
 - Dictionary, 92
 - List, 90
 - łańcuchów, 379
 - SortedList, 91
 - SortedDictionary, 92
 - kolory
 - komponentów, 120
 - paska postępu, 236
 - w .NET, 30
 - komentarze, 359
 - kompilacja
 - dwustopniowa, 47
 - kodu pośredniego, 48
 - komponentu, 219
 - warunkowa, 82
 - kompilator Visual Basic, 49
 - komponent, 193
 - BackgroundWorker, 268
 - ComboBox1, 396
 - DataGridView, 307
 - DataSet, 302, 304
 - FileListBox, 194, 224, 228
 - ListBox, 171, 179
 - NumericUpDown, 313
 - OsobyBindingSource, 312
 - PrintDocument, 137
 - SerialPort, 193
 - SoundPlayer, 145
 - StatusStrip, 123
 - TextBox, 129, 130
 - Timer, 146, 151
 - kompozycja Klasyczny
 - Windows, 228
 - komunikat, 412
 - o wyjątku, 43
 - WM_NCMOUSEMOVE, 413
 - komunikaty Windows, 407
 - konfiguracja
 - timera, 152
 - konfigurowanie
 - formy, 183
 - interfejsu, 392
 - komponentu DataSet, 302, 325
 - kontrolki DataSet, 326
 - paska stanu, 123
 - konstrukcja
 - SyncLock .. End SyncLock, 282
 - Try..Catch, 314
 - Try..Catch..End Try, 130
 - Using..End Using, 130
 - konstruktor
 - klasy opisującej formę, 31
 - obiektu Form1, 167
 - konstruktory domyślne, 25, 97
 - kontrola typów, 53
 - kontrolka
 - ColorProgressBar, 230, 236
 - DataGridView, 346
 - DataSet, 326
 - FileListBox, 194, 223–228
 - panel, 19
 - PrintDialog, 137
 - ProgressBar, 228, 230, 235
 - suwak, 19
 - kontrolki ActiveX, 193
 - kontynuacja polecenia, 49
 - konwersja typu, 74
 - kopiowanie pliku tekstowego, 129
- L**
- leniwe inicjowanie zmiennych, 65
 - liczba instancji aplikacji, 34
 - LIFO, last in, first out 92
 - LINQ, Language Integrated Query, 317
 - LINQ to DataSet, 315, 325–328
 - LINQ to Objects, 315, 317
 - LINQ to SQL, 315, 335, 337
 - LINQ to XML, 315, 371, 373
 - LINQ to XSD, 380
 - lista, 90
 - argumentów metody, 250
 - dysków logicznych, 266
 - klas, 247
 - komunikatów, 412
 - metod, 248
 - osób, 318
 - parametryczna, 85, 92
 - procesów, 168
 - przenoszonych plików, 181
 - rozszerzeń, 392
 - typów, 247
 - z błędami, 90
- Ł**
- ładowanie
 - dynamiczne bibliotek, 245
 - statyczne bibliotek, 243
 - łańcuch, 59
 - łączenie
 - danych, 349
 - zbiorów, 322
- M**
- malowanie kontrolki, 229
 - mapowanie oparte na atrybutach, 337
 - mechanizm
 - PInvoke, 399, 403
 - późnego wiązania, 239
 - reflection, 255
 - menu
 - Edycja, 134
 - główne, 124
 - kontekstowe, 148, 189
 - metatypy, 95
 - metoda, 95
 - Abort, 274
 - Append, 62
 - Application.Run, 33, 257
 - Array.Sort, 111
 - Assembly.GetType, 248
 - Button1_Click, 41
 - Color.FromArgb, 30
 - ColorProgressBar_Paint, 233
 - CompareTo, 106, 110
 - Console.Out.WriteLine, 260
 - Console.WriteLine, 261
 - CreateDatabase, 341
 - DeleteDatabase, 341
 - DoDragDrop, 171, 172, 176
 - Element, 374
 - Equals, 104
 - Form.Dispose, 25
 - Form1.Dispose, 25

- metoda
- Form1_FormClosed, 361
 - Form1_Paint, 158
 - Get, 242
 - GetHashCode, 103
 - GetProcesses, 167
 - GetType, 59
 - GetValueOrDefault, 74
 - Graphics.DrawImage, 159
 - inicjująca przenoszenie, 177, 179
 - InitializeComponents, 25
 - Invoke, 253
 - Join, 280
 - kopiująca tekst, 129
 - ListBox1_DoubleClick, 213, 215
 - Main, 26, 33, 34, 259
 - Marshal.SizeOf, 52
 - Me.Controls.Add, 25
 - MessageBox.Show, 49, 69, 130, 346
 - Monte Carlo, 268
 - Object.GetType, 245
 - OnClick, 211
 - OnValueChanged, 235
 - OsobyTableAdapter.Fill, 313
 - Parallel.For, 292
 - PerformStep, 234
 - pobierająca listę plików, 197
 - pobierająca opis, 394
 - PobierzZawartoscKatalogu, 200, 202
 - podziału długich linii, 142, 143
 - PostMessage, 407
 - pozwalająca na wybór pliku, 131
 - Program.Main, 263
 - ReadEndElement, 362
 - ReadStartElement, 362
 - ReadToEnd, 130
 - Refresh, 216
 - SendMessage, 407
 - Serialize, 366
 - Show, 242, 249
 - ShowMessage, 70
 - Single, 320
 - SubmitChanges, 338, 340, 341
 - Thread.Abort, 273
 - Thread.Join, 288
 - ToDouble, 99
 - ToString, 99
 - TrackBar1_ValueChanged, 27, 30
 - Type.GetMethod, 250
 - upraszczająca ułamek, 99
 - WndProc, 407, 411
 - WriteComment, 361
 - WriteStartDocument, 361
 - zamykająca aplikację, 149
 - zapisująca tekst, 132
 - zarządzająca drukowaniem, 137, 144
 - zwracająca wartość, 68
- metody
- klasy String, 60
 - menu Edycja, 135
 - niewspółdzielone, 254
 - przeciążone, 66
 - publiczne, 249
 - rozszerzające, extension methods, 114
 - rozszerzające LINQ, 318
 - statyczne, 253
 - uruchamiane z menu, 191
 - współdzielone, 253
- metody zdarzeniowe, 27, 65
- dwukrotne kliknięcie listy, 203
 - kliknięcie
 - Czcionka, 134
 - Drukuj, 138
 - Kolor tła, 134
 - Zapisz jako, 133
 - zamknięcie aplikacji, 126
- Microsoft Access, 297
- modyfikacja metody zdarzeniowej, 157
- modyfikator
- Async, 295
 - ByRef, 404
 - Friend, 25
 - Overrides, 25
 - Private, 97
 - Public, 97
 - Shared, 98
 - WithEvents, 25, 27
- modyfikowanie
- danych, 323, 355
 - pliku XML, 379
 - rekordu, 339
- monitor komunikatów, 411
- N**
- narzędzie
- O/R Designer, 342
 - uzupełniania kodu, 118
- nawiasy
- klamrowe, 298
 - kwadratowe, 233, 242
- O**
- obiekt, 95
- NotifyIcon, 148
 - rejestr, 393
- obliczenia sekwencyjne, 291
- obsługa
- aplikacji okienkowej, 32
 - menu kontekstowego, 190
 - wyjatków, 42
- odczytywanie
- atrybutów elementów, 365, 374
 - danych z pliku XML, 361
 - danych z rejestru, 383
 - nazwy elementu, 374
 - ustawień, 155
- odśmieciacz, garbage collector, 25, 73
- odświeżanie
- formy, 156
 - komponentu, 216
- okno
- Call Stack, 42
 - Choose Toolbox Items, 223
 - Locals, 40
 - podpowiedzi, 149
 - Server Explorer, 304
 - ustawień wydruku, 139
 - Watch, 40
 - własności, Properties Window, 18, 232
- OLE DB, 297
- opakowanie, 74
- opcja
- Explicit, 12, 53
 - Explicite, 212
 - Infer, 12, 55
 - Strict, 53, 212
 - Show All Files, 26
 - Startup form, 33
 - Strict, 12, 255
 - ustalania typu, 345
- operator
- &=, 62
 - Join, 322, 349
 - Like, 62
 - łączenia łańcuchów &, 49
 - New, 25, 73
 - Order By, 332
 - przypisania =, 50
 - TypeOf.Is, 59
 - Where, 332

- operatory
 - arytmetyczne, 56, 101
 - dzielenia, 55
 - konwersji, 104
 - logiczne i bitowe, 58
 - porównania, 57, 103
 - VB, 56
 - opis biblioteki, 222
 - osobny wątek, 270
 - ostrzeżenie, 42
- P**
- Parallel Extensions, 267
 - Parallel LINQ, 267
 - pasek postępu, 233
 - pełna kompilacja, 335
 - pętla, 78
 - Do..Loop While, 91
 - For, 79, 186
 - For Each, 79, 180
 - For zrównoleżona, 291
 - foreach, 88
 - Parallel.For, 267
 - platforma .NET, 47, 381, 399
 - platforma aplikacji, application framework, 33
 - plik
 - AdresyDataSet.xsd, 303
 - App.config, 153, 154
 - Application.Designer.vb, 26
 - Application.myapp, 26
 - AssemblyInfo.vb, 221
 - Class1.vb, 242
 - FileListBox.dll, 224
 - FileListBox.vb, 196, 207
 - Form1.Designer.vb, 21
 - Form1.resx, 21
 - Form1.vb, 17, 209
 - Kolor.exe, 28
 - Settings.Designer.vb, 26
 - Telefony.dbml, 342
 - user.config, 156
 - pliki
 - .acddb, 326
 - .dll, 239
 - .exe, 48, 239
 - .mdb, 326
 - .mdf, 326
 - .sdf, 326
 - *.Designer.vb, 26
 - dźwiękowe, 151
 - konfiguracyjne, 17
 - projektu, 17, 21
 - rozwiązania i projektu, 17
 - wykonywalne, 239
 - ustawienia.xml, 357, 364, 368
 - pobieranie danych, 353
 - pobieranie danych z tabeli, 338
 - podgląd wydruku, 140, 141
 - podglądanie wartości zmiennych, 40
 - podkatalog System32, 153
 - podklucze, 389
 - podmiana pliku .dll, 243
 - podniesienie flagi, 307
 - podokno
 - Properties, 19, 127
 - Solution Explorer, 32
 - Toolbox, 19
 - podprogram, subroutine, 66
 - podzespoły, assembly, 252
 - podział linii, 143
 - poła prywatne, 97, 196
 - pole, 95
 - pole typu string, 263
 - polecenie
 - Add Watch, 41
 - AddHandler, 70
 - anonym.GetType().FullName, 118
 - Application.Exit(), 127
 - Close, 126, 127
 - Delete, 299
 - Insert, 299
 - Select, 298
 - tworzące instancję klasy, 130
 - położenie okna, 387
 - potwierdzenie zamknięcia aplikacji, 127
 - późne wiązanie, 255
 - prezentacja katalogu, 199
 - prezentowanie danych, 346
 - priorytet wątku, 276
 - procedura składowana, 353–356
 - proces, 166, 275
 - profil użytkownika, 264
 - program Reflection, 58
 - programowanie
 - asynchroniczne, 267, 294
 - obiektywne, 24
 - wizualne, 16, 26
 - współbieżne, 47
 - zdarzeniowe, 126
 - programowanie poleceń, 126
 - Czcionka, 134
 - Drukuj..., 139, 144
 - Kolor tła, 134
 - menu Edycja, 134
 - odczytywanie tekstu, 129
 - Otwórz..., 131
 - Podgląd wydruku, 141
 - potwierdzenie zamknięcia, 127
 - Ustawienia strony..., 136, 140
 - zamknięcie okna, 127
 - Zapisz jako..., 132
 - projekt
 - aplikacji konsolowej, 257
 - bazy danych, 326
 - biblioteki DLL, 240
 - biblioteki kontrolki, 219
 - typu Windows Forms Application, 163
 - AdoNet_SqlServer, 299
 - DragAndDrop, 170
 - DywanGraficzny, 156
 - Ekran powitalny, 145
 - elektroniczna kukułka, 145
 - FileListBoxDemo, 194
 - ikiTekstowe, 225
 - Kolorowy pasek postępu, 228
 - Kolory, 29
 - Mandelbrot, 163
 - Procesy, 166
 - prosty edytor, 119
 - TypyPlikow, 391
 - projektowanie interfejsu, 120
 - ikona okna, 121
 - menu główne, 124
 - nazwa okna, 121
 - pasek stanu, 123
 - ustawienia, 122
 - projektowanie nowego typu, 95
 - prywatne pola klasy, 24
 - przechowywanie tekstu, 129
 - wyjątków, 43, 45, 80
 - przeciąganie elementów, 171, 228
 - przeciągnij i upuść, 169
 - przeciążanie metody, overload, 67
 - przeglądarka
 - plików tekstowych, 224, 226
 - zespołów, 252
 - przekazywanie argumentów
 - przez referencje, 69
 - przez wartości, 69
 - przekazywanie parametru, 283
 - przekroczenie zakresu, 37
 - przenoszenie danych do pliku XML, 376, 377
 - elementu, 174–177
 - formy, 188

- przenoszenie
 - okna, 188
 - plików, 181
 - wielu elementów, 179
 - przepelnienie, 45
 - przerywanie pętli, 293
 - przestrzeń nazw
 - System.Threading, 271
 - System.Collections, 85
 - System.Collections.Generics, 90
 - System.ComponentModel, 205
 - System.Data.Linq, 336
 - System.Data.Linq.Mapping, 336
 - System.Drawing, 366
 - System.Media, 145
 - System.Messaging, 407
 - System.Threading.Tasks, 292
 - System.Windows.Forms, 22, 25
 - przezroczystość, 183, 186
 - przypisywanie metody
 - do zdarzeń, 29
 - pseudonimy, 92
 - pudełkowanie, boxing, 59, 75
 - pula wątków, 285, 286
 - punkt wejścia, entry point, 257
 - pusta forma, 17
 - pusta referencja, 74
- R**
- referencja, 25
 - rejestr, 391
 - rejestr systemowy, 383
 - relacja rodzic – dziecko, 352
 - relacje, Associations, 351
 - rozpoznawanie typów, 58, 239
 - rozszerzenia plików, 393
 - rozszerzenie
 - AsEnumerable, 332
 - CopyToDataTable, 330
 - String, 114
 - UsunApostrof, 115
 - z delegacją, 115
 - rozwiązanie, solution, 17
 - różnica między
 - #If a If, 82
 - strukturami i klasami, 73
 - rysowanie
 - dywanu, 159
 - paska postępu, 233
 - zbioru Mandelbrota, 160
 - rzutowanie wartości, 54
 - rzutowanie zawężające,
 - narrowing, 53
- S**
- schowek, 134
 - sekcja
 - Case Else, 77
 - Catch, 80
 - Finally, 81
 - Try, 44, 80
 - serializacja do pliku XML, 367
 - siatka DataGridView, 305
 - składowane zapytanie, 355
 - skojarzenia plików, 389
 - słowniki, 85, 92
 - słowo kluczowe
 - ALTER, 353
 - AS, 353
 - ByRef, 69
 - Case, 77
 - Class, 72, 96
 - CREATE, 353
 - Delegate, 70
 - Event, 70
 - Function, 68
 - Get, 242
 - Handles, 27
 - Inherits, 22
 - Me, 26, 31
 - Nothing, 74, 86
 - null, 74
 - Overrides, 99
 - Partial, 22
 - RaiseEvent, 212
 - REM, 49
 - Resources, 26
 - Return, 68
 - Step, 233
 - Structure, 72, 96
 - Sub, 66
 - Try, 79
 - With, 87, 101, 116
 - sortowanie, 89, 106, 111, 312, 332
 - splash screen, 145
 - sprawdzanie danych, 314
 - SQL Server, 297
 - stała DEBUG, 82
 - stałe liczbowe, 52
 - stałe statyczne, 97
 - stan połączenia, 304
 - sterta, 73
 - stos, stack, 42, 73, 92
 - strongly typed, 336, 344
 - struktura, 72, 95
 - Color, 29
 - Nullable(Of Integer), 74
 - Okno, 366
 - System.Int32, 50
 - SystemColor, 29
 - typu anonimowego, 118
 - Ulamek, 96
 - zapytania LINQ, 319
 - strumień
 - błędów, 260
 - Console.Error, 260
 - StringReader, 138
 - symbol preprocesora DEBUG, 82
 - synchroniczne wykonywanie kodu, 294
 - synchronizacja wątków, 289
 - szablon
 - biblioteki Class Library, 224
 - biblioteki kontrolek, 220
 - Console Application, 258
- Ś**
- ścieżka
 - do katalogu, 205, 385
 - do plików, 133, 199
 - śledzenie
 - aplikacji, 278
 - wykonywania programu, 37
 - zmian w katalogu, 217
 - środowisko Visual Studio, 15
- T**
- tablica znaków, 406
 - tablice, 84, 85
 - deklaracja, 85
 - gromadzenie referencji, 87
 - jednowymiarowe, 87
 - rozmiar, 85
 - usuwanie, 86
 - wielowymiarowe, 87
 - zawartość, 86
 - technologia
 - LINQ, 47, 315
 - PLINQ, 267
 - testowanie
 - funkcji WinAPI, 409
 - metody zdarzeniowej, 27
 - operatorów arytmetycznych, 102

- par mieszanych, 114
 - zdarzeń, 214
 - struktury, 98
 - timer, 149, 152
 - TPL, Task Parallel Library, 267
 - Transact SQL, 298
 - transformacja klasy w strukturę, 97
 - tryb
 - debugowania, 38
 - edycji, 39
 - pojedynczego wątku, 33
 - tworzenie
 - automatyczne interfejsu użytkownika, 347
 - biblioteki DLL, 240
 - dotatkowego wątku, 145
 - formularzy, 309
 - interfejsu, 19
 - klasy reprezentującej dane, 343
 - komponentu, 194
 - kopii zapasowej, 371
 - metody zdarzeniowej, 126
 - obiektów, 373
 - plików XML, 361
 - pliku bazy danych, 300
 - pliku XML, 371
 - projektu aplikacji, 16, 120
 - przycisków, 87
 - puli wątków, 286
 - tabeli w SQL Server, 336
 - wątku, 271
 - typ
 - DataRow, 329
 - Integer, 41
 - Lazy, 65
 - List, 129
 - Long, 41
 - String, 61, 129
 - StringBuilder, 61
 - System.Int64, 404
 - T, 110
 - Ulamek, 105
 - ULong, 42
 - Variant, 12
 - typy
 - anonimowe, 117, 333
 - liczbowe, 89
 - ogólne, generic types, 107
 - parametryczne, 107
 - proste, 50
 - referencyjne, 25, 72, 95
 - wartościowe, 72, 95
 - wyliczeniowe, 63, 264, 402
 - zdefiniowane automatycznie, 329
- ## U
- uchwyt okna, 408
 - ukrywanie paska stanu, 128
 - upuszczanie elementu, 173
 - uruchamianie metod
 - niewspółdzielonych, 254
 - z biblioteki, 253
 - uruchomione procesy, 166
 - ustawienia
 - aplikacji, application settings, 153, 388
 - drukowania, 140
 - kompilatora, 56
 - usuwanie klucza, 388
 - usypianie wątku, 272
 - użycie przestrzeni nazw, 262
- ## V
- Visual Basic, 48
 - Visual Basic 2010 Express, 15
 - Visual Studio 2010 Ultimate, 15
- ## W
- wartości domyślne pól, 109
 - wartości zmiennych, 40
 - wartość, value, 92
 - wątek, thread, 267
 - działanie w tle, 275
 - kończenie pracy, 280
 - przerywanie działania, 273
 - przesyłanie danych, 283
 - sygnalizowanie zakończenia, 288
 - synchronizowanie, 282, 289
 - usypianie, 272
 - wstrzymywanie, 274
 - wznawianie, 274
 - zmienianie priorytetu, 276
 - wczytywanie
 - biblioteki do pamięci, 246
 - danych, 369
 - obrazu z pliku, 185
 - pliku do kontrolki, 227
 - pliku tekstowego, 225
 - wiązanie zdarzeń, 210
 - widok projektowania, 21, 148
 - widok projektowania formy, 154
 - wielkość liter, 24, 207
 - wielowątkowość, 268
 - wizytówka, 147
 - własności
 - komponentów, 30
 - kontrolki ColorProgressBar, 230
 - własności pola, 205
 - własność, 95
 - Anchor, 20
 - BackColor, 31
 - BalloonTipTitle, 150
 - Checked, 128
 - e.KeyChar, 32
 - Filter, 131, 132
 - Filtr, 207
 - Interval, 151
 - Licznik, 101
 - Lines, 130
 - Location, 25
 - Me.Handle, 410
 - Mianownik, 101
 - NotifyFilter, 219
 - Path, 219
 - Parent, 25
 - ŚcieżkaKatalogu, 206
 - TabIndex, 25
 - tylko do odczytu, 208
 - Visible, 25, 128
 - wnoskowanie typów kolekcji, 55
 - wolne miejsce na dysku, 404
 - WPF, Windows Presentation Foundation, 12, 119
 - wybór
 - czcionki, 134
 - kontrolki, 223
 - pliku, 131
 - wyciek pamięci, 73
 - wyjątek
 - NullReferenceException, 87
 - System.ArgumentOutOfRangeException, 91
 - wyjątki, 42, 79, 130
 - wyjątki typu Exception, 80
 - wrażenia lambda, 71, 116
 - wyświetlenie komunikatu, 242
 - wywołanie
 - funkcji z biblioteki DLL, 244
 - metody, 66
 - metody zdarzeniowej, 31
 - wyzwalacz, trigger, 235
 - wzorce łańcuchów, 63

X

XML, Extensible Markup Language, 357

Z

zablokowanie programu, 282
 zakładka Form1.vb [Design], 17
 zamykanie aplikacji, 187
 zapis
 do pliku, 159
 do pliku XML, 359
 zapisywanie
 tekstu, 132
 ustawień, 155
 zapytania LINQ, 315, 378
 zbiory
 kontrolerek, Toolbox, 17
 projektów, 17
 zbiór
 Julii, 165
 Mandelbrota, 160, 165

zdarzenia klasy Form, 412

zdarzenie

 Click, 172, 209, 403
 ComboBox1.SelectedIndexChanged, 314
 domyślne komponentu, 126
 DoubleClick, 203
 DragOver, 173
 DwukrotneKliknieciePliku, 214
 Form.MouseDown, 116
 FormClose, 127
 FormClosing, 127
 MouseClicked, 172
 MouseDown, 172, 227
 PagePrint, 139
 Paint, 156, 185
 PrintPage, 137
 SelectedIndexChanged, 210, 396
 ValueChanged, 26, 29, 235
 ZmianaKatalogu, 212

zmiana Class na Structure, 96
 zmiana przestrzeni nazw
 biblioteki, 243
 zmienna środowiskowa PATH, 153
 zmienne
 referencyjne, 72
 środowiskowe, 264
 wartościowe, 73
 znak
 |, 132
 *, 353
 Backspace, 61
 Escape, 61
 kontynuacji, 49
 końca linii, 61
 nowej linii, 130
 tabulacji, 61
 znaki ASCII, 32
 znikanie okna, 186
 zwracanie wartości, 68, 403

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄŻKI**
- 3. ZBIERAJ PROWIZJĘ**

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

VISUAL BASIC to język programowania o bardzo długiej tradycji. Kiedy kilka lat temu przeniesiono go na platformę **.NET**, zyskał nowe możliwości rozwoju. Dziś chętnie używają go osoby, które potrzebują wygodnego narzędzia, pozwalającego szybko stworzyć aplikację do prezentacji i analizy konkretnych danych. Jasna, niezbyt skomplikowana składnia, doskonała biblioteka gotowych kontrolki i bezkolizyjna współpraca z bazą danych to najważniejsze atuty **VB.NET**.

Jeśli jesteś początkującym użytkownikiem Visual Basic na platformie **.NET**, trafiłeś w dziesiątkę. Tu znajdziesz podstawy języka oraz wszystkie funkcje potrzebne do swobodnego projektowania aplikacji — bez nadmiernego wchodzenia w szczegóły. Zorientujesz się, jak działa środowisko Visual Studio i jak za pomocą jego narzędzi tworzyć aplikacje **VB.NET**. Dowiesz się więcej o debugowaniu kodu, programowaniu zorientowanym obiektowo i używaniu kontrolki, zarówno tych z biblioteki Windows Forms, jak i zaprojektowanych osobiście. Odkryjesz, jak można połączyć aplikację z bazą danych i jak wykorzystać technologie systemu Windows. To wszystko, czego Ci trzeba!

- Środowisko Visual Studio i język Visual Basic
- Przegląd komponentów biblioteki Windows Forms
- Projektowanie zorientowane obiektowo i projektowanie kontrolki
- Zarządzane biblioteki DLL i mechanizm rozpoznawania typów
- Aplikacje konsolowe i informacje o systemie
- Wątki, równoległa pętla Parallel.For oraz programowanie asynchroniczne (Async/Await)
- Podstawy ADO.NET i SQL Server
- Wprowadzenie do zapytań LINQ na przykładzie kolekcji (LINQ to Objects)
- LINQ to DataSet i LINQ to SQL
- Odczytywanie i zapisywanie danych w plikach XML
- Rejestr systemu Windows, mechanizm PInvoke i komunikaty Windows



helion.pl
księgarnia
internetowa

Nr katalogowy: 9699



Księgarnia internetowa
<http://helion.pl>



Zamówienia telefoniczne:

0 801 339900



0 601 339900



Helion

Sprawdź najnowsze promocje:

• <http://helion.pl/promocje>

Książki najchętniej czytane:

• <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

• <http://helion.pl/nowosci>

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
<http://helion.pl>

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

ISBN 978-83-246-4898-6



9 788324 164898 6

Cena 79,00 zł

Informatyka w najlepszym wydaniu