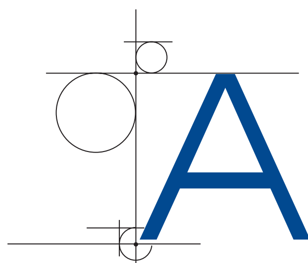


Język C#

A.1. Aplikacje konsolowe w języku C#

A.1.1. Wprowadzenie

Język C# (wymawiamy: „C sharp”) jest językiem przeznaczonym do tworzenia aplikacji, które działają w środowisku .NET Framework. Jest to prosty, nowoczesny i uniwersalny język obiektowy umożliwiający tworzenie aplikacji dla systemu Windows, aplikacji internetowych oraz aplikacji mobilnych. Implementacją języka C# na platformie .NET jest Visual C#.

Zintegrowane środowisko programistyczne Visual Studio obsługuje język Visual C# oraz dostarcza dla niego takich narzędzi, jak edytor kodu, kompilator, szablony projektów, debuger i inne. .NET Framework zapewnia dostęp do wielu usług, systemów operacyjnych i innych narzędzi, a także zawiera biblioteki i środowisko uruchomieniowe do uruchamiania aplikacji napisanych za pomocą Visual C#.

Do tworzenia aplikacji w trybie pulpitu wykorzystywany jest pakiet Visual Studio 2012 Express for Desktop. Pobrany i zainstalowany pakiet można uruchomić. Jeżeli został zainstalowany polski pakiet językowy, język polski powinien być dostępny automatycznie. Jeżeli tak się nie stało lub chcemy zmienić język domyślny, należy w menu wybrać *Narzędzia/Opcje* (*Tools/Options*) i w otwartym oknie dialogowym wybrać opcję *Ustawienia międzynarodowe* (*International Settings*), a następnie w polu *Język* (*Language*) ustawić język, który powinien obowiązywać dla interfejsu użytkownika.

UWAGA

W podręczniku do programowania w języku C# została wykorzystana wersja Visual Studio 2012 Express for Desktop oraz Visual Studio 2012 Express for Web.

A.1.2. Środowisko pracy

Rozpoczynając pracę z językiem C# w środowisku Visual Studio, należy utworzyć nowy projekt. W tym celu z zakładki *Plik* trzeba wybrać opcję *Nowyprojekt* i w otwartym oknie dla *Visual C#* wskazać opcję *Aplikacja konsoli*. Opcja ta pozwala tworzyć aplikacje konsolowe i na potrzeby nauki języka C# jest wystarczająca.

W chwili rozpoczęcia pracy z aplikacją tworzony jest automatycznie plik *Program.cs*. Zawiera on standardowy kod rozpoczynający tworzenie programu. W zależności od tego, jaki typ projektu zostanie wybrany, Visual C# automatycznie generuje standardowy dla tego typu kod. Przykład pokazuje kod wygenerowany automatycznie przez Visual C# dla *Aplikacja konsoli*.

Przykład A.1

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Program1
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Wygenerowany kod zawiera definicje używanych w programie przestrzeni nazw. Pod deklaracjami przestrzeni znajduje się przestrzeń robocza. Zaczyna się ona od słowa `namespace` i nazwy programu. Zawarty w nawiasach klamrowych kod stanowi zaczątek tworzonego programu. Widoczna jest w nim jedna klasa `Program` oraz statyczna metoda `Main()`. Tworzony przez programistę kod programu powinien zostać umieszczony w nawiasach klamrowych `{...}` po poleceniu `static void Main(string[] args)`.

Kompilowanie programu

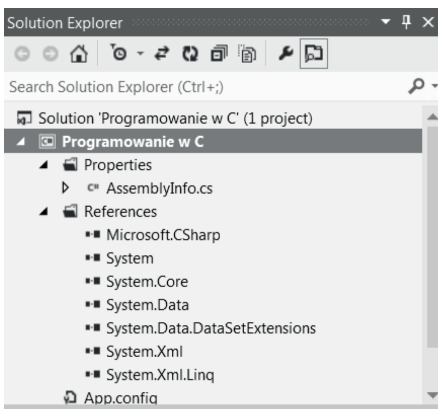
Utworzony kod można skompilować i uruchomić, naciskając klawisz *F5* (ze śledzeniem błędów) lub *Ctrl+F5* (bez śledzenia błędów). Inną metodą uruchomienia programu jest kliknięcie na pasku *Standardowy* zielonej strzałki z napisem *Rozpocznij* lub wybranie z menu *Debuguj* opcji *Start Debugging* lub *Start Without Debugging*. Podczas kompilowania programu kompilator sprawdza, czy kod źródłowy nie zawiera błędów. Jeżeli kod

nie zawiera błędów, instrukcje programu są wykonywane w takiej kolejności, w jakiej zostały zapisane.

Struktura projektu

Aplikacje .NET tworzone są w ramach tak zwanych projektów. Projekt w C# to pojedynczy program. Jednak możliwe jest tworzenie elementów nadrzędnych w stosunku do projektu, zawierających kilka projektów. Ten nadrzędny element można traktować jak system informatyczny. W technologii .NET przyjmuje on nazwę *Rozwiązanie* (*Solution*).

Z prawej strony głównego okna pakietu VS wyświetlany jest panel *Eksplorator rozwiązania* (*Solution Explorer* — rysunek A.1), który umożliwia nawigację po strukturze tworzonego rozwiązania. Jeżeli panel nie jest wyświetlany, należy wybrać w menu *Widok/Eksplorator rozwiązania*.



Rysunek A.1. Solution Explorer

W panelu tym można dodawać oraz usuwać pliki, nawigować po strukturze rozwiązania i wybierać pliki, których zawartość powinna zostać wyświetlona. Tworzony system może zawierać wiele połączonych ze sobą projektów. Nowy projekt w ramach tego samego rozwiązania można dodać, posługując się panelem *Eksplorator rozwiązania*.

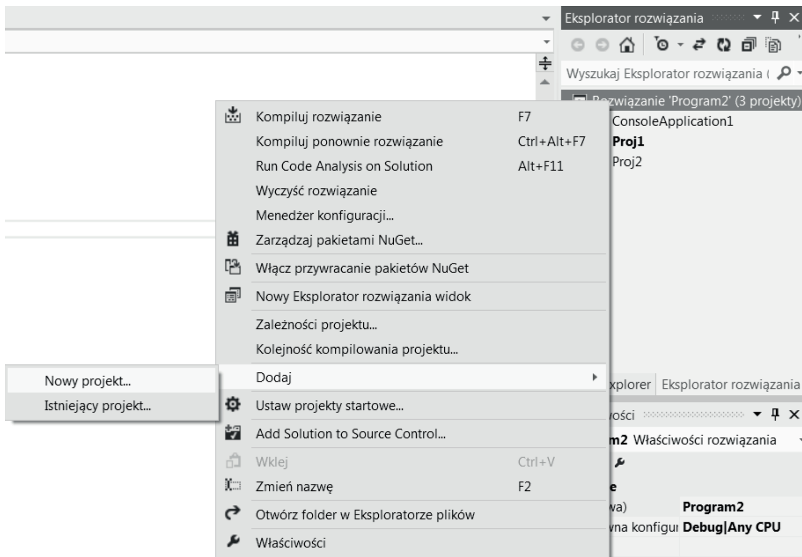
Poniżej panelu *Eksplorator rozwiązania* znajduje się panel *Właściwości*, który wyświetla właściwości aktualnie zaznaczonego elementu.

W dolnej części okna znajduje się panel *Dane wyjściowe*, który wyświetla listę wykonanych zadań i błędów występujących podczas wykonywania programu.

Dodatkowe panele można włączyć po wybraniu z menu opcji *Widok*.

Przykład A.2

W panelu *Eksplorator rozwiązania* klikamy prawym przyciskiem myszy nazwę całego rozwiązania i z wyświetlonego menu kontekstowego wybieramy opcję *Dodaj/Nowy projekt…* (rysunek A.2). W otwartym oknie wybieramy opcję *Aplikacja konsoli* i wprowadzamy nazwę projektu, np. `Projekt1`.



Rysunek A.2. Dodawanie nowego projektu w panelu Eksplorator rozwiązania

W ramach bieżącego rozwiązania zostanie utworzony nowy projekt.

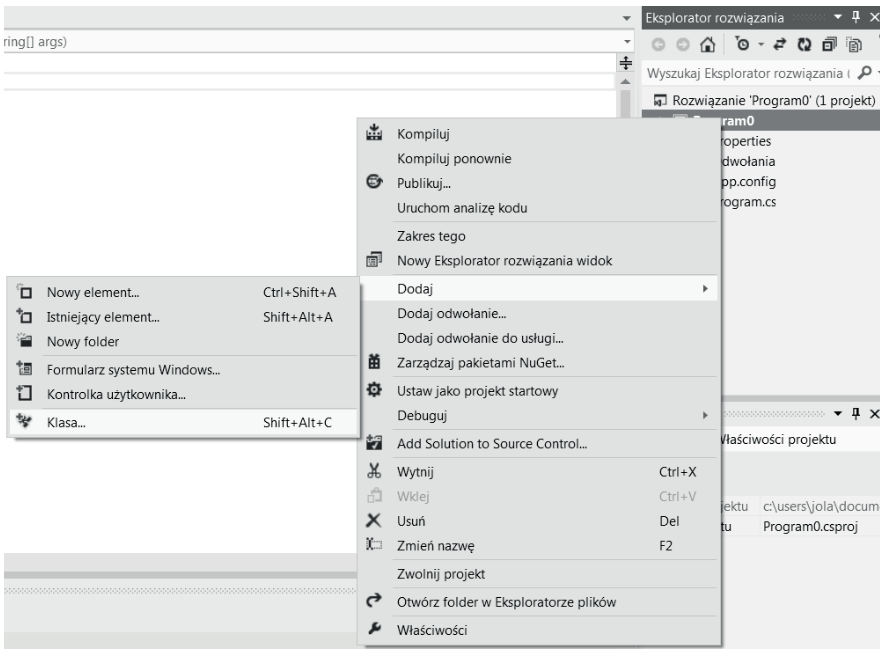
Korzystając z panelu *Eksplorator rozwiązania*, do utworzonego projektu można dodawać nowe elementy.

Przykład A.3

Do projektu zostanie dodany nowy plik zawierający definicję klasy. W panelu *Eksplorator rozwiązania* klikamy prawym przyciskiem myszy nazwę tworzonego projektu. Z wyświetlonego menu kontekstowego wybieramy opcję *Dodaj/Klasa…* (rysunek A.3). W otwartym oknie wprowadzamy nazwę pliku, np. *NowaKlasa.cs*.

Po utworzeniu pliku z nową klasą w edytorze kodu źródłowego możemy zobaczyć jego zawartość.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Program2
{
    class NowaKlasa
    {
    }
}
```



Rysunek A.3. Dodawanie nowych elementów w panelu Eksplorator rozwiązania

Kod, który został wygenerowany, występuje w definicji każdej klasy. Utworzony plik ma taką samą nazwę jak nazwa klasy. Powinniśmy przestrzegać zasady, że utworzony plik zawiera klasę o takiej samej nazwie jak nazwa pliku oraz że w jednym pliku znajduje się definicja jednej klasy.

Zapisanie zmian wprowadzonych w projekcie następuje po wybraniu z menu *Plik/Zapisz wszystko*. Przy ponownym uruchamianiu Visual Studio wystarczy wybrać opcję *Otwórz projekt…* i z listy wybrać projekt, z którym zamierzamy pracować.

A.1.3. Przestrzenie nazw

Biblioteka klas, dostępna w .NET Framework, składa się z klas, typów i stałych. Elementy biblioteki są pogrupowane w przestrzenie nazw, które tworzą strukturę hierarchiczną. Dzięki temu mogą istnieć dwie klasy o tej samej nazwie, jednak pod warunkiem, że zostały zadeklarowane w różnych przestrzeniach nazw. Nazwy przestrzeni nazw zawsze zaczynają się od słowa `System` lub `Microsoft`, na przykład: `System.IO`, `System.Console`. Przestrzenie nazw wprowadzają pewną organizację, hierarchię i porządek w bibliotekach klas (tabela A.1).

Tabela A.1. Główne przestrzenie nazw platformy .NET

Przestrzeń nazw	Opis
`System`	Definiuje podstawowe klasy oraz jest podstawą wielu podrzędnych przestrzeni
`System.Collections`	Definiuje klasy będące kolekcjami
`System.IO`	Definiuje klasy do zarządzania strumieniami wejścia i wyjścia i do tworzenia oraz odczytu plików
`System.Security`	Definiuje klasy związane z bezpieczeństwem oraz kryptografią
`System.Net`	Definiuje klasy umożliwiające programowanie sieciowe oraz dostęp do takich usług, jak DNS czy HTTP
`System.Data`	Definiuje klasy związane z dostępem do baz danych
`System.Web`	Definiuje klasy oraz inne przestrzenie związane z dostępem do sieci, usługami sieciowymi oraz protokołem HTTP
`System.Windows.Forms`	Definiuje klasy związane z biblioteką Windows Forms oraz projektowaniem wizualnym

Włączenie danej przestrzeni nazw następuje przy użyciu słowa kluczowego `using`.

A.2. Składnia języka C#

Język C# jest językiem obiektowym, więc każdy program napisany w tym języku jest klasą lub kolekcją klas. Wszystkie zmienne i metody są zawarte w definicji klasy. Klasę definiuje się za pomocą słowa kluczowego `class`, po którym występuje nazwa klasy. Każdy blok kodu opisującego klasę musi być umieszczony między nawiasami klamrowymi `{...}`.

Instrukcje

W języku C# każda instrukcja musi być zakończona średnikiem. Natomiast kod może być pisany bez podziału na linie, jednak dla jego czytelności należy zachowywać układ proponowany przez środowisko języka C# z podziałem na linie i z zastosowaniem wcięć. W języku C# istotna jest wielkość liter. Wyrazy `Komputer` oraz `komputer` to dla C# dwa różne słowa.

A.2.1. Metoda Main

Obowiązkowym elementem każdego programu jest metoda `Main`. To od niej program rozpoczyna swe działanie i na niej kończy. Kod najprostszego programu może mieć postać podaną w przykładzie niżej.

Przykład A.4

```
class Prog
{
    static void Main(string[] args)
    {
        System.Console.WriteLine("Mój pierwszy program");
    }
}
```

W podanym przykładzie klasa `Prog` zawiera metodę `Main`, która wywołuje metodę `WriteLine`. Metoda ta jest składową klasy `Console` znajdującej się w przestrzeni nazw `System`. Aby za każdym razem przy odwoływaniu się do obiektów lub metod nie pisać całej ścieżki z uwzględnieniem przestrzeni nazw, można na początku programu umieścić instrukcję `using`, np. `using System`. Określi ona, z jakiej przestrzeni nazw będą używane klasy.

Przykład A.5

```
using System;
class Prog
{
    static void Main(string[] args)
    {
        Console.WriteLine("Mój pierwszy program");
    }
}
```

Metoda `Main` jest domyślnie deklarowana jako statyczna i prywatna. Ponieważ punkt wejścia reprezentowany przez metodę `Main` nie jest publiczny, inny proces nie będzie mógł uruchomić aplikacji. Słowo `static` przed metodą `Main` oznacza, że metoda może działać, mimo że nie zostały utworzone obiekty klasy `Prog`. Słowo `void` oznacza, że metoda `Main` nie zwraca żadnego wyniku. Klasa `Prog` to klasa główna programu, ponieważ zawiera metodę `Main`. Wykonanie programu rozpocznie się od pierwszej instrukcji wewnątrz metody `Main`.

`System` to przestrzeń nazw, które można traktować jako zbiór zdefiniowanych klas. `Console` to klasa wbudowana umieszczona w przestrzeni `System`, w której znajduje się definicja między innymi metody `WriteLine()`. Metoda `WriteLine()` to metoda, którą należy wykonać, aby wyświetlić na ekranie tekst. Teksty muszą być umieszczone w cudzysłowach.

Klasy, które będziemy tworzyć, możemy umieszczać we własnych przestrzeniach. Przestrzenie, klasy, zmienne i metody tworzą w pewien sposób hierarchię: przestrzenie zawierają klasy, te natomiast zmienne i metody.

A.2.2. Komentarze

W języku C# występują trzy rodzaje komentarzy. Pierwszy z nich to wstawienie podwójnego znak ukośnika //. Tekst znajdujący się za tymi znakami nie jest interpretowany przez kompilator. Jest to komentarz jednej linii. Wiele linii komentarza zapisujemy między znakami /* oraz */. Trzeci rodzaj komentarza to komentarz dokumentacji kodu w XML. Tworzony jest on po wpisaniu trzech znaków ukośnika ///. Służy do generowania dokumentacji w kodzie XML. Za pomocą komentarzy można opisywać tworzone w programie instrukcje. W trakcie kompilacji komentarze są usuwane z pliku wynikowego.

Przykład A.6

Komentarz składający się z wielu linii:

```
/* początek
tekst
tekst
koniec komentarza */
```

Komentarz jednoliniowy:

```
// początek i koniec komentarza
```

Komentarz dla dokumentacji:

```
/// <summary>
/// This class does...
/// </summary>
class Program {
...
}
```

A.2.3. Zmienne

Deklarując zmienną, należy nadać jej unikatową nazwę i określić jej typ. Jeżeli kilka zmiennych jest tego samego typu, mogą zostać zadeklarowane razem. Nazwa zmiennej może być dowolna, ale musi spełniać następujące warunki:

- musi zaczynać się od litery lub znaku podkreślenia,
- może składać się z liter, cyfr i znaku podkreślenia,
- w nazwie rozróżniane są małe i duże litery,
- w nazwach można stosować polskie litery.

Zmienna przed jej użyciem musi zostać zainicjowana. Deklaracja zmiennej ma postać:

```
typ_zmiennej nazwa_zmiennej [= wartość];
```

Przykład A.7

```
class Prog
{
    static void Main(string[] args)
    {
        string Nazw, Imie, Log;
    }
}
```

W podanym przykładzie zostały utworzone zmienne `Nazw`, `Imie`, `Log`. Są one typu `string`. Mogą zostać zadeklarowane również tak:

```
string Nazw;
string Imie;
string Log;
```

Zadeklarowane w ten sposób zmienne nie mają przypisanych wartości. Do przypisania wartości jest używany operator =.

Przykład A.8

```
class Prog
{
    static void Main(string[] args)
    {
        string Nazw = "Kowalski";
    }
}
```

A.2.4. Stałe

Stałe to identyfikatory, których wartości nie ulegają zmianie. Stałe, podobnie jak zmienne, są identyfikowane przez nazwę. Nazwa musi spełniać podobne warunki jak nazwa zmiennej. Deklaracja stałych ma postać:

```
const typ_stałej nazwa_stałej = wartość;
```

A.2.5. Typy danych

W języku C# każda zmienna ma swój typ. Określa on, jakiego rodzaju wartości mogą zostać przypisane do zmiennej. Typy zmiennych dzielą się na:

- typy proste,
- typy referencyjne.

Proste typy danych

Tabela A.2 przedstawia proste typy danych języka C#.

Tabela A.2. Proste typy danych

Typ danych	Opis
byte	Liczba od 0 do 255
sbyte	Liczba od –128 do 127
short	Liczba od –32 768 do 32 767
int	Liczba całkowita
uint	Liczba od 0 do 4 294 967 295
long	Liczba całkowita długa
ulong	Liczba długa, wartości dodatnie
float	Liczba pojedynczej precyzji
double	Liczba podwójnej precyzji
decimal	Liczba dziesiętna
char	Pojedynczy znak
string	Łańcuch znaków typu char
bool	Wartość true lub false

Typy definiowane przez użytkownika

Typy definiowane przez użytkownika mogą być używane w taki sam sposób jak typy proste (wbudowane). Do typów definiowanych przez użytkownika należą typy wyliczeniowe i struktury.

Typ wyliczeniowy

Typ wyliczeniowy jest zbiorem stałych o określonych wartościach. Definiując taki typ, trzeba określić zbiór dopuszczalnych wartości całkowitych. Każdej z tych wartości należy przyporządkować niepowtarzalne identyfikatory. Zmienna typu wyliczeniowego może przechowywać tylko wartości z tego zbioru.

Składnia typu wyliczeniowego ma postać:

```
enum nazwa_typu [:typ_bazowy]
{
etykieta_stałej_1 [= wartość_1]
[,etykieta_stałej_2 [= wartość_2]
[, ... etykieta_stałej_n [= wartość_n]]]]
};
```

Typem bazowym dla typu wyliczeniowego są liczby całkowite. Jeżeli nie zostanie podany ten typ, to domyślnie będzie przyjęty typ `int`. Jeżeli etykietom stałej nie przypisze się wartości, zostaną im nadane kolejne wartości, począwszy od `0`.

Przykład A.9

```
enum DniTyg
{
Poniedziałek,
Wtorek,
Środa,
Czwartek,
Piątek,
Sobota,
Niedziela
};
```

Żeby zadeklarować zmienną typu wyliczeniowego, wystarczy określić nazwę typu i nazwę zmiennej.

Przykład A.10

```
dzien = DniTyg.Sobota;
Console.WriteLine(DniTyg.Wtorek);
```

W podanym przykładzie zmiennej `dzien` została przypisana wartość typu wyliczeniowego `DniTyg`, a następnie została wykonana instrukcja wyświetlająca na ekranie napis *Wtorek*.

Struktura

Struktura to typ definiowany przez użytkownika. Może składać się z wielu innych typów. Może zawierać konstruktory, stałe, pola, metody, właściwości, operatory. Ten typ jest stosowany do definiowania obiektów. Do jego definiowania używane jest słowo kluczowe `struct`.

Typy referencyjne

Zmienne typu referencyjnego nie przechowują wartości, lecz odwołania do danych. Do typów referencyjnych należą:

- typy klas (`class`),
- typy interfejsów (`interface`),
- typy delegacji (`delegate`),
- typy tablicowe (`array`).

A.2.6. Operatory

Operatory można pogrupować ze względu na ich przeznaczenie. Podstawowe operatory zostały przedstawione w tabeli A.3.

Tabela A.3. Podstawowe operatory

Kategoria	Operatory
Arytmetyczne	`+`, `-`, `*`, `/`, `%`
Porównania	`==`, `!=`, `<`, `>`, `<=`, `>=`
Bitowe	`&`, `|`, `^`, `~`
Logiczne	`&&`, `||`, `!`
Przypisania	`=`, `+=`, `-=`, `*=`, `/=`, `%=`

A.2.7. Operacje na konsoli — klasa System.Console

Klasa `Console` jest jedną z wielu klas zdefiniowanych w przestrzeni nazw `System`. Jej zadaniem jest obsługa operacji wejścia-wyjścia, czyli wyświetlanie informacji na ekranie lub odczytywanie tekstu wpisanego przez użytkownika. Klasa `Console` posiada wiele metod statycznych, co oznacza, że nie muszą być tworzone obiekty klasy `Console`, aby używać jej metod. Do najważniejszych metod należą poznane już `WriteLine()` oraz `ReadLine()`. Wybrane metody i właściwości klasy `System.Console` zostały pokazane w tabeli A.4.

Tabela A.4. Wybrane metody i właściwości klasy System.Console

Metody i właściwości	Opis
`Beep()`	Wydaje dźwięk
`Clear()`	Kasuje zawartość konsoli
`SetCursorPosition`	Ustawia pozycję kursora w oknie konsoli
`Write()`	Wysyła tekst do konsoli
`WriteLine()`	Wysyła tekst do konsoli wraz ze znakiem nowej linii
`ReadLine()`	Czyta informacje z klawiatury, dopóki nie pojawi się znak nowej linii; zwraca wartość typu `string`
`Read()`	Czyta z klawiatury pojedyncze znaki, zwraca wartość typu `char`
`ReadKey()`	Odczytuje znaki z konsoli (klawiatury)

Metody i właściwości	Opis
BackgroundColor	Ustawia kolor tła
ForegroundColor	Ustawia kolor tekstu
ResetColor	Ustawia domyślny kolor tła i tekstu
SetWindowSize	Ustawia wielkość okna konsoli

Przykład A.11

```
namespace Program1
{
    class Program
    {
        static void Main(string[] args)
        {
            // ustaw rozmiar okna
            Console.SetWindowSize(60, 30);
            // ustaw położenie tekstu
            Console.SetCursorPosition(10, 10);
            Console.WriteLine("Witaj w moim programie!");
            Console.ReadLine();
            Console.Clear();
            Console.Beep();
        }
    }
}
```

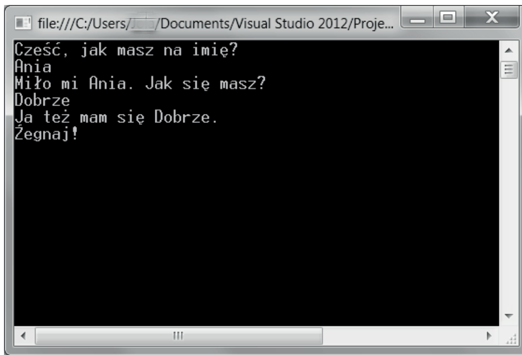
W podanym przykładzie `Console` to klasa, w której zdefiniowane zostały metody obsługi konsoli.

Przykład A.12

```
namespace Program2
{
    class Program
    {
        static void Main(string[] args)
        {
        Console.WriteLine("Cześć, jak masz na imię?");
```

```
            string imie; // deklaracja zmiennej
            imie = Console.ReadLine(); // pobranie tekstu wpisanego przez użytkownika
            Console.WriteLine("Miło mi " + imie + ". Jak się masz?");
            string masz;
            masz = Console.ReadLine();
            Console.WriteLine("Ja też mam się " + masz + ".");
            Console.WriteLine("Żegnaj!");
            Console.ReadLine();
            }
        }
    }
```

Podany przykład pokazuje wykorzystanie metod klasy `Console` do budowania interakcji z użytkownikiem. Wynik wykonania kodu został przedstawiony na rysunku A.4.



Rysunek A.4. Wykorzystanie metod klasy `Console` do budowania interakcji z użytkownikiem

Przykład A.13

Podany w przykładzie A.12 kod zostanie zmodyfikowany przez utworzenie klasy `Jakie_imie`; dla utworzonej klasy będą zdefiniowane dwie metody: `Czytaj()` i `Napisz()`.

```
class Jakie_imie
{
    string imie;
    public void Czytaj()
    {
        System.Console.WriteLine("Cześć, jak masz na imię?");
        imie = System.Console.ReadLine();
```

```
    }
    public void Napisz()
    {
        System.Console.WriteLine("Miło mi " + imie + ". Jak się masz?");
    }
}
class Start
{
    static void Main()
    {
        Jakie_imie obiektImie = new Jakie_imie();
        obiektImie.Czytaj();
        obiektImie.Napisz();
        System.Console.ReadLine();
    }
}
```

W klasie `Jakie_imie` zostały zdefiniowane dwie metody: `Czytaj()` i `Napisz()`. Metody zostały poprzedzone słowem `public`, co oznacza, że są dostępne dla innych klas, np. dla klasy `Start`. Klasa `Start` zawiera metodę `Main`, w której został utworzony obiekt klasy `Jakie_imie`. Dla utworzonego obiektu zostały wywołane metody `Czytaj()` i `Napisz()`.

Znak nowej linii

Wywołanie metody `WriteLine()` spowoduje przejście do nowego wiersza. Ale ten sam efekt można osiągnąć, wstawiając w łańcuch tekstowy znak nowej linii. Znak ten ma postać `\n`.

Przykład A.14

```
class Start
{
    static void Main()
    {
        System.Console.WriteLine("Jan Kowalski, \nul. Krótka 23, \n23-456 Wligosz");
        System.Console.ReadLine();
    }
}
```

Zadanie A.1

W środowisku Visual Studio dla języka C# utwórz nowy projekt pod nazwą `Nowy użytkownik`. Następnie napisz program, który będzie pytał użytkownika o nazwisko, imię oraz adres i dla potwierdzenia poprawności wczytania wyświetlał te dane na ekranie.

A.3. Obiektowość języka C#

Język obiektowy, którym jest C#, umożliwia tworzenie nowych typów danych. Nowy typ danych tworzy się poprzez zdefiniowanie klasy.

A.3.1. Klasy i obiekty

Klasa to zbiór powiązanych ze sobą metod oraz zmiennych (nazywanych polami). Klasa definiuje wszystkie swoje elementy, czyli określa, jak jej metody działają oraz co oznaczają poszczególne jej pola. Aby korzystać z klasy, trzeba utworzyć jej instancję, czyli obiekt. Klasa jest definicją, natomiast obiekt jest jej fizycznym reprezentantem. W języku C# wszystkie operacje wykonywane są na obiektach. Można utworzyć wiele obiektów będących instancjami tej samej klasy. Każdy z tych obiektów posiada swoje pola oraz metody.

Sposób deklarowania klasy:

```
class nazwa_klasy
{ ...
definicje_zmiennych_oraz_metod
...
}
```

Sposób tworzenia obiektu:

```
nazwa_klasy nazwa_obiektu = new konstruktor()
```

Przykład A.15

```
class Osoba
{
public void Pisz()
{
Console.WriteLine("Osoba");
}
}
```

Utworzenie obiektu klasy:

```
Osoba o1 = new Osoba();
```

```
//Wywołanie metody klasy:
o1.Pisz();
```

Konstruktor klasy to specjalna metoda, która określa, jakie czynności zostaną wykonane w momencie tworzenia obiektu danej klasy. Podczas jego użycia można przypisać zmiennym wartości początkowe. W każdej klasie jest zdefiniowany domyślny konstruktor, który można zastąpić własnym. Konstruktor nazywa się tak samo jak klasa, w której został utworzony. W jednej klasie można utworzyć kilka konstruktorów i używać ich w zależności od potrzeb.

Przykład A.16

```
class Osoba
{
string nazwisko;
public Osoba(string nazw)
{
nazwisko = nazw;
}
public void Pisz()
{
Console.WriteLine("Nazwisko:  " + nazwisko);
}
}
Osoba o1 = new Osoba("Nowak");
o1.Pisz();
```

Konstruktor został zdefiniowany tu:

```
public Osoba(string nazw)
{
nazwisko = nazw;
}
```

Definicję konstruktora powinno poprzedzać słowo `public`, ponieważ będzie on używany wewnątrz innych klas. W podanym przykładzie parametr, jaki przyjmuje konstruktor, to `nazw`. W treści konstruktora zmiennej `nazwisko` zostaje przypisana wartość podana jako parametr.

Destruktor to metoda określająca czynności wykonywane podczas usuwania obiektu z pamięci. Język C# jest językiem zarządzanym, co oznacza, że platforma .NET dba o usuwanie z pamięci niepotrzebnych obiektów. Destruktor ma postać:

```
~nazwa_klasy {...}
```

Przykład A.17

```
class Osoba
{
~Osoba()
{
Console.WriteLine("Usunięte");
}
}
Osoba o1 = new Osoba("Nowak");
o1.Pisz();
o1 = null;
```

A.3.2. Własności

Własności klasy służą do zarządzania polami (zmiennymi) definiowanej klasy. Definiują one sposób dostępu do zmiennych obiektu z zewnątrz. Własności nie przechowują danych — określają tylko miejsce ich przechowywania.

Ogólna deklaracja własności ma postać:

```
specyfikator_dostępu typ nazwa_własności
{
get
{...}
set
{...}
}
```

`Get` i `Set` są to tak zwane *akcesory*. Akcesor `Set` ustala miejsce, do którego ma być przypisana wartość parametru `value`. Akcesor `Get` zwraca określoną wartość.

Przykład A.18

```
{
...
string nazwisko;
public string Nazwisko
{
get
{
```

```
return nazwisko.ToUpper();
}
set
{
if (value == "Nowak" || value == "Kowal")
nazwisko = value;
else
Console.WriteLine("Musisz podać nazwisko Nowak lub Kowal");
}
}
}
Osoba o1 = new Osoba("Nowak");
o1.Pisz();
o1.Nazwisko = "Górka";
o1.Nazwisko = "Kowal";
o1.Pisz();
Console.ReadLine();
```

A.3.3. Przeciążanie funkcji

Jeśli zdefiniowana metoda przyjmuje w zależności od potrzeby różną liczbę parametrów lub różne typy parametrów, to jest to przeciążanie definicji metody. Polega ona na definiowaniu kilku metod o tej samej nazwie posiadających różne parametry lub różną ich liczbę. Środowisko języka C# decyduje samo na podstawie kontekstu wywołania metody, którą jej wersję należy wykonać.

Przykład A.19

```
static void Dodaj(int liczba1, int liczba 2)
{
Console.WriteLine(liczba 1+ liczba 2);
} static void Dodaj(string liczba 1, string liczba 2)
{
Console.WriteLine(liczba 1 + liczba 2);
} static void Dodaj(int liczba 1, int liczba 2, int liczba 3)
{
Console.WriteLine(liczba 1 + liczba 2 + liczba 3);
}
```

```
Dodaj(5, 7);
Dodaj("Jan", "Kowalski");
Dodaj(3,7,4);
```

A.3.4. Zasięg widoczności

Zasięg widoczności definiuje, w jakim zakresie będą widziane tworzone zmienne, metody oraz klasy. W praktyce oznacza to określenie, z jakiego miejsca w kodzie będzie można się odwołać do wybranych zmiennych czy obiektów.

Dostępne są następujące zakresy widoczności (inaczej *specyfikatory dostępu*):

- `public` — metody, zmienne oraz klasa są dostępne z dowolnego miejsca w kodzie,
- `protected` — metody, zmienne oraz klasa dostępne są tylko z kodu klasy lub klasy po niej dziedziczącej,
- `internal` — metody, zmienne oraz klasa dostępne są tylko z kodu tego samego projektu,
- `protected internal` — tak samo jak `internal`, z wyjątkiem klas dziedziczących, które mają dostęp nawet z innych projektów.
- `private` — dostęp tylko z aktualnej klasy.

Przykład A.20

```
public int numer;
public static void Dodaj(int liczba 1, int liczba 2)
public class Osoba
```

A.3.5. Klasy, pola i metody statyczne

Czasami przydatne może być używanie klasy bądź jej elementów (pola, metody) bez tworzenia jej instancji. Służy do tego słowo kluczowe `static`. Nie można tworzyć obiektów klas statycznych, zatem służą one głównie do grupowania zmiennych oraz metod. Metody statyczne oraz pola statyczne można wykorzystywać bez tworzenia obiektów danej klasy. Pole statyczne danej klasy może być tylko jedno i jest dostępne poprzez klasę. Klasy niestatyczne mogą posiadać metody i pola statyczne, natomiast klasy statyczne nie mogą posiadać pól i metod niestatycznych.

Przykład A.21

```
public static class Dodawanie
{
public static string name = "nazwa";
public static int Dodaj(int par1, int par2)
{
```

```
return par1 + par2;
}
}
...
Console.WriteLine(Dodawanie.Dodaj(3, 8));
Console.ReadLine();
```

A.3.6. Dziedziczenie

Kluczowym aspektem programowania obiektowego jest mechanizm dziedziczenia. Polega on na tym, że podczas definiowania nowej klasy można wskazać klasę nadrzędną (rodzica), po której tworzona klasa ma dziedziczyć określone metody i pola.

Przykład A.22

```
public class Książki
{
public void Opis()
{
Console.WriteLine("To jest książka!");
}
}
public class Albumy : Książki
{
}
```

W przykładzie zdefiniowano klasę `Książki`, która posiada jedną publiczną metodę `Opis`. Metoda ta wyświetla tekst *To jest książka!*. Druga zdefiniowana klasa, `Albumy`, dziedziczy po klasie `Książki` (`public class Albumy : Ksiazki`). Oznacza to, że metoda `Opis` dostępna będzie również w klasie `Albumy`.

Jeśli do klasy `Albumy` zostanie dodana metoda:

```
public override void Opis()
{
Console.WriteLine("To jest album!");
}
```

to w nowej klasie zostanie nadpisana metoda klasy nadrzędnej `Opis` z jej nową wersją (słowo kluczowe `override`).

A.4. Kompilacja i debugowanie

A.4.1. Kompilacja

Kompilacja to proces, który tłumaczy program napisany w języku C# na język zrozumiały dla komputera. Kompilację projektu w VS można wykonać, wybierając z menu *Kompilacja*. Na liście opcji dostępne są pozycje: *Kompiluj rozwiązanie* (kompilacja zostanie wykonana od nowa), *Kompiluj ponownie rozwiązanie* (nastąpi kompilowanie tylko wprowadzonych zmian) i *Wyczyść rozwiązanie* (nastąpi usunięcie wszystkich plików kompilacji).

Przed rozpoczęciem kompilacji można ustawić jej parametry, wybierając właściwości całego projektu. W tym celu należy w menu wybrać *Projekt/Właściwości/Nazwa projektu*. Otworzy się okno, w którym z lewej strony znajduje się lista opcji wyboru konfiguracji. Po wybraniu opcji *Aplikacja* można ustalić nazwę aplikacji, platformę docelową i typ wyjściowy aplikacji. Klikając przycisk *Informacje o zestawie...*, można zapisać informacje o wersji aplikacji. Można też wybrać ikonę, która zostanie przypisana do pliku programu. Po wybraniu opcji *Kompilacja* można określić w polu *Platforma docelowa* architekturę platformy, na którą przeznaczona jest aplikacja. Do wyboru mamy 32-bitową, 64-bitową oraz dowolną (*Any CPU*). Wybierając opcję *Publikuj*, można wskazać miejsce publikacji projektu (np. na serwerze FTP) oraz określić numer wersji publikacji. Parametry kompilacji zapisujemy, klikając prawym przyciskiem myszy ich zakładkę i wybierając z menu polecenie *Zapisz zaznaczone elementy*. Po tych działaniach okno właściwości można zamknąć i skompilować program z bieżącymi ustawieniami, wybierając z menu *Kompilacja/Kompiluj rozwiązanie*.

A.4.2. Debugowanie

Debugowanie to sposób śledzenia wykonywania instrukcji programu. Można wybrać debugowanie krokowe lub ciągłe. Debugowanie krokowe to ręczne śledzenie poprawności wykonywania kodu. Pozwala ono znaleźć błędy w kodzie i poprawić je. Po wybraniu w menu opcji *Debuguj* mamy do wyboru *Start Debugging*, czyli uruchomienie programu z debugowaniem, *Start Without Debugging*, czyli uruchomienie programu bez debugowania, i *Attach to Process...*, czyli przechodzenie do kolejnych linijek kodu. W kodzie programu można ustawić tzw. *breakpoint*, czyli punkt, w którym ciągłe debugowanie zostanie zatrzymane i nastąpi przejście do debugowania krokowego. W celu ustawienia tego punktu należy w odpowiednim miejscu kodu programu kliknąć jego lewy margines. Czerwony punkt wskaże miejsce zatrzymania. Do tego miejsca kod programu zostanie wykonany w sposób ciągły, dalsze wykonywanie programu będzie się odbywało w trybie krokowym. Po uruchomieniu wykonywania programu zatrzyma się on we wskazanym miejscu. Naciskając klawisz *F10*, możemy wywoływać wykonanie kolejnych instrukcji w trybie krokowym. Na dole, w panelu *Local*, po wykonaniu każdego kroku wyświetlane będą zawartości poszczególnych zmiennych. Śledząc je, można zlokalizować przyczynę błędu. Punkt *breakpoint* można w dowolnym momencie usunąć, ponownie go klikając. Po sprawdzeniu poprawności działania aplikacji można zapisać ją w pliku docelowym.

Aby otrzymać przekompilowany plik gotowego projektu, najlepiej wyczyścić kompilacje, które były przeprowadzone wcześniej (wybierając z menu *Kompilacja/Wyczyść rozwiązanie*), i wykonać ponowną kompilację, wybierając *Kompilacja/Kompiluj rozwiązanie*. Następnie należy odnaleźć skompilowany plik projektu. Domyślnie projekty zapisywane są w bibliotece dokumentów, dlatego tam należy szukać pliku wynikowego — trzeba wybrać *Moje Dokumenty/Visual Studio 2012/Projects/Nazwa projektu/Nazwa projektu/bin/Debug*. Tutaj powinien znajdować się plik *exe* utworzonej aplikacji. Jego uruchomienie spowoduje wykonanie programu poza środowiskiem programistycznym. Można to sprawdzić, kopiując plik np. na pulpit i uruchamiając go. Tak przygotowany plik można udostępnić użytkownikom. Do prawidłowego działania aplikacji wymagane jest zainstalowanie platformy .NET Framework w wersji, jaka została ustalona w parametrach tworzonego projektu.

A.5. Aplikacje internetowe w języku C#

Użycie Visual Studio Express For Web umożliwia sprawne i intuicyjne tworzenie aplikacji internetowych, dla których podstawą architektoniczną są platforma .NET Framework (z ASP i ADO .NET), serwer IIS oraz Microsoft SQL Server.

Tworzenie dynamicznych stron ASP.NET odbywa się w C# w sposób wizualny, za pomocą formularza oraz zestawu kontrolek. Właściwe elementy (komponenty) są umieszczane na stronie, a środowisko Visual Studio w tle generuje odpowiedni kod HTML.

A.5.1. Formularze Web Forms

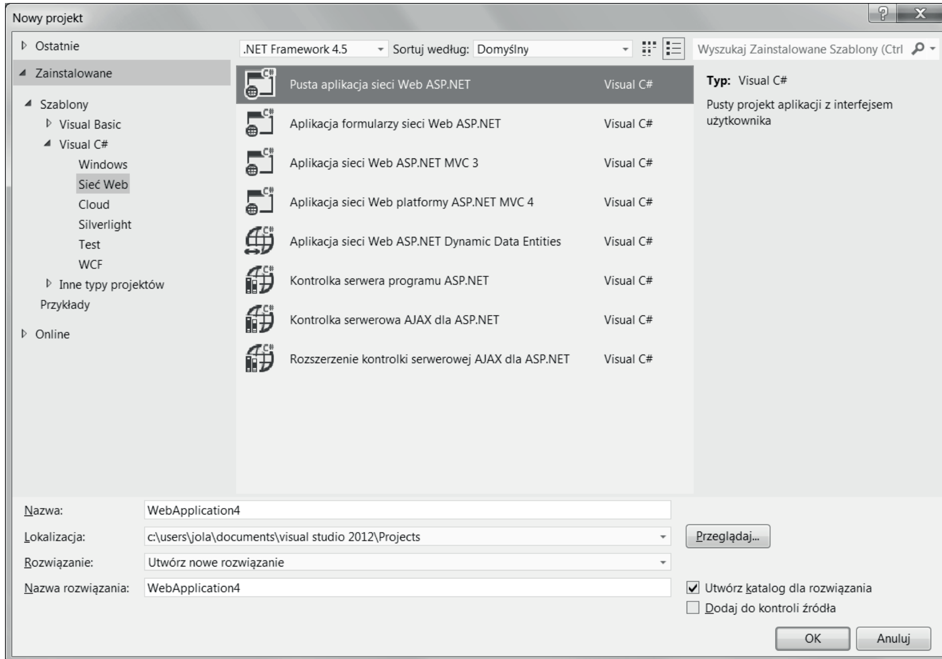
Przestrzeń nazw `System.Windows.Forms` to podstawowa biblioteka służąca do projektowania wizualnego. Dołączana jest do każdego projektu typu `WinForms`. Zawiera podstawowe klasy obsługi aplikacji, ale również wszystkie klasy komponentów. Każdy komponent jest jednocześnie klasą, która zawiera (jak każda klasa) właściwości, metody, zdarzenia i struktury.

Klasy biblioteki `WinForms` można podzielić na kilka kategorii:

- *Komponenty* — są to niewidoczne elementy programu, bloki działające w tle, które mogą mieć duży wpływ na działanie aplikacji, lecz nie są częścią jej interfejsu.
- *Kontrolki* — to komponenty wizualne. Dzięki nim można tworzyć interfejs swojego programu. Do tej kategorii można zaliczyć przyciski, listy rozwijane, pola edycyjne.
- *Elementy pozycjonowania* — to komponenty, które umożliwiają zarządzanie elementami interfejsu, np. komponent umożliwiający tworzenie zakładek, komponent `Panel`, który grupuje inne kontrolki.
- *Menu* i *paski narzędziowe* — to komponenty odpowiedzialne za tworzenie menu czy pasków narzędziowych.
- *Okna dialogowe* — to ukryte komponenty, które odpowiadają za wyświetlanie okien dialogowych, np. *Otwórz*, *Zapisz jako*, *Drukuj*.

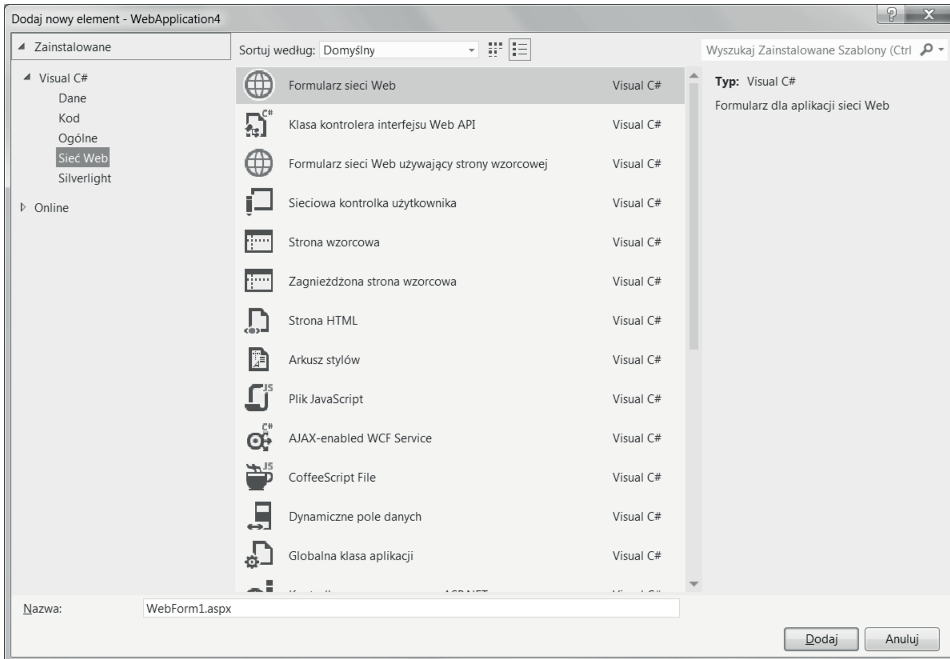
A.5.2. Projektowanie formularzy

Aby utworzyć prostą aplikację internetową, należy uruchomić Visual Studio Express For Web i wybrać z menu głównego opcję *Plik/Nowy projekt*. W oknie dialogowym *Nowy Projekt* należy zaznaczyć opcję *Visual C#/Sieć Web* i wybrać szablon *Pusta aplikacja sieci Web ASP.NET* (rysunek A.5). Można jeszcze wpisać nazwę tworzonego projektu. Zostanie utworzona pusta aplikacja internetowa.



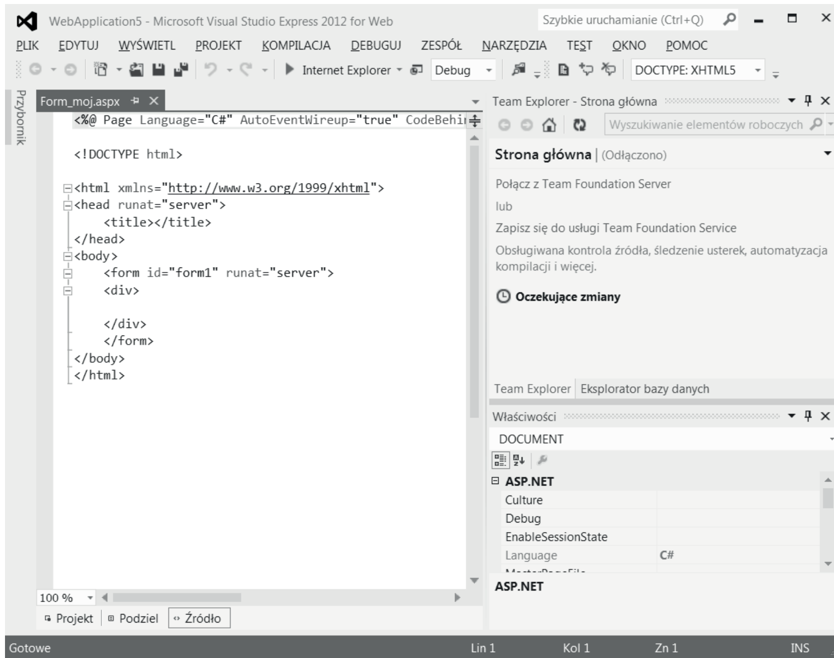
Rysunek A.5. Okno Nowy Projekt aplikacji Visual Studio

Pierwszym krokiem projektowania aplikacji może być umieszczenie w niej formularza. W tym celu należy z menu głównego wybrać *Projekt/Dodaj nowy element*. Wyświetli się okno listy szablonów. W oknie z lewej strony z dostępnej listy należy wybrać *Visual C#/Sieć Web*, a następnie z listy szablonów *Formularz sieci Web* (rysunek A.6). W dolnej części okna trzeba jeszcze wpisać nazwę tworzonego pliku, np. *Form1*, i kliknąć przycisk *OK*.



Rysunek A.6. Okno listy szablonów aplikacji Visual Studio

W wyniku zostanie utworzony plik o podanej nazwie z rozszerzeniem *.aspx* (np. *Form1.aspx*). Dodatkowo powstaną plik kodu ukrytego o takiej samej nazwie, z rozszerzeniem *.aspx.cs* (np. *Form1.aspx.cs*), oraz plik z rozszerzeniem *.aspx.desinger.cs*, w którym zostanie zapisany kod generowany automatycznie przez Visual Studio. Utworzony plik *Form1.aspx* można wyświetlać w jednym z trzech widoków. Domyślny jest widok źródła (*Źródło*), przedstawiający kod HTML. Drugi widok projektu (*Projekt*) przedstawia stronę w taki sposób, w jaki będzie ona wyświetlana w przeglądarce. Trzeci widok podzielony (*Podziel*) pokazuje stronę jednocześnie w widoku kodu HTML oraz tak, jak będzie ona wyświetlana w przeglądarce. Przełączanie między widokami jest dostępne w dolnej części okna edytora (rysunek A.7).



Rysunek A.7. Okno tworzenia aplikacji

A.5.3. Kontrolki formularza

W kodzie HTML, który został automatycznie wygenerowany, za pomocą znacznika `<form>` został utworzony formularz w postaci:

```
<form id="form1" runat="server">
```

Występujący tu atrybut `runat="server"` jest traktowany jako kontrolka serwerowa. Jeżeli znacznik zawiera taki atrybut, to znaczy, że powinien zostać wykonany przez ASP.NET Framework na serwerze. Atrybut `runat="server"` nie wchodzi w skład standardu HTML, dlatego ASP.NET usuwa ten atrybut z kodu HTML przed przesłaniem go do przeglądarki.

Wewnątrz formularza zostały umieszczone dwa znaczniki: otwierający `<div>` i zamykający `</div>`. Między tymi znacznikami można umieszczać własny kod opisujący stronę. Mogą to być teksty lub różne elementy formularza.

Przykład A.23

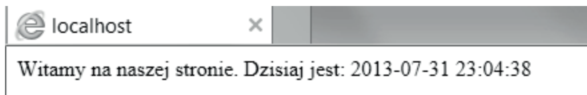
```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.
cs" Inherits="WebApplication15.WebForm1" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head runat="server">
    <title></title>
</head>
<body>
<form id="form1" runat="server">
<div>
Witamy na naszej stronie. Dzisiaj jest:  <%= DateTime.Now.ToString() %>
</div>
</form>
</body>
</html>
```

Przykład pokazuje kod wygenerowany automatycznie, uzupełniony o tekst wpisany między znacznikami `<div>` i `</div>`. Umieszczone w kodzie znaczniki `<%` i `%>` oznaczają wstawienie kodu C#. Znak = umieszczony zaraz po znaczniku otwierającym poinformuje ASP.NET, że w kodzie znajduje się wyrażenie, które powinno zostać przetworzone. Naciśnięcie klawisza *F5* spowoduje uruchomienie kodu, przekompilowanie fragmentu zapisanego w języku C# i wysłanie wyniku do przeglądarki (rysunek A.8).



Rysunek A.8. Kod zapisany w języku C# został przekompilowany i wysłany do przeglądarki

Dodawanie kontrolek

Kontrolki serwerowe mogą być umieszczane w formularzu na trzy sposoby:

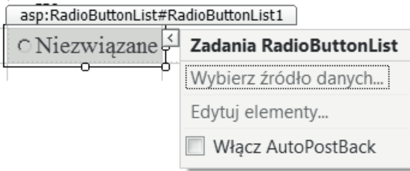
- przez wpisanie odpowiedniego kodu w pliku *.aspx*,
- przez przeciągnięcie z *Przybornika* odpowiedniej ikony kontrolki,
- przez napisanie kodu, który je doda w trakcie działania programu.

Przykład A.24

W formularzu zostaną wyświetlone trzy przyciski opcji umożliwiające użytkownikowi wybór wykształcenia (*podstawowe*, *średnie*, *wyższe*). Można to uzyskać, przeciągając grupę przycisków opcji z panelu *Przybornik* znajdującego się z lewej strony okna edytowania.

Jeżeli panel *Przybornik* nie jest widoczny w oknie, można go wyświetlić, wybierając z menu głównego *Wyświetl/Przybornik*. Kontrolki widoczne w panelu *Przybornik* można uporządkować, klikając w obszarze tego panelu prawym przyciskiem myszy i wybierając z wyświetlonego menu opcję *Sortuj elementy alfabetycznie*.

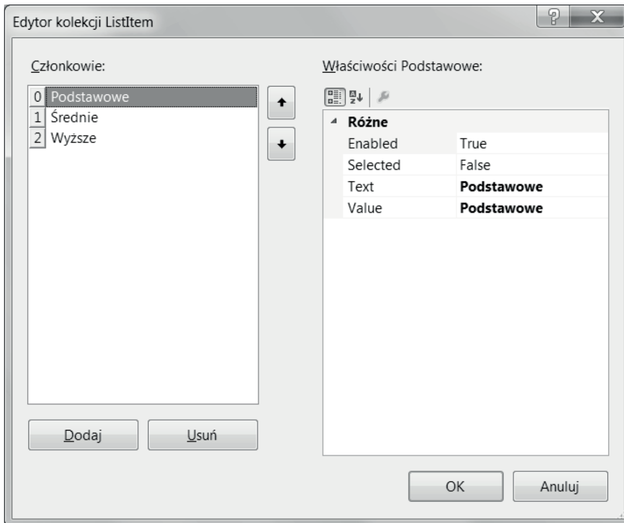
Wybieramy w panelu *Przybornik* kontrolkę `RadioButtonList` i przeciągamy ją i upuszczamy w odpowiednim miejscu w kodzie (między znacznikami `<div>`). Przechodzimy do widoku *Projekt* i zaznaczamy utworzony element. Przy zaznaczeniu elementu pojawi się *inteligentny znacznik* (ang. *smart tag*, rysunek A.9), który umożliwia określenie źródła danych (np. można powiązać przycisk z danymi w bazie danych) lub samodzielne wpisanie danych.



Rysunek A.9. Inteligentny znacznik kontrolki

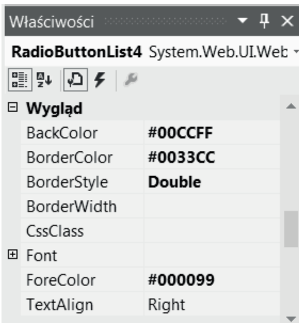
Po wybraniu inteligentnego znacznika i kliknięciu opcji *Edytuj elementy* wyświetli się okno *Edytor kolekcji ListItem*. Można w nim zdefiniować grupę opcji wyboru. Robimy to poprzez dodanie kolejnych przycisków opcji (klikając przycisk *Dodaj*). Dodajemy opcje przycisków: *Podstawowe*, *Średnie*, *Wyższe*.

Każdy przycisk będzie miał domyślną nazwę `ListItem`. Natomiast wartości i ich opisy należy wprowadzić samodzielnie. Można też określić, które z trzech pól będzie początkowo zaznaczone (rysunek A.10).



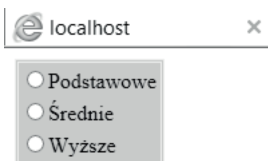
Rysunek A.10. Okno Edytora kolekcji ListItem

Po utworzeniu grupy opcji klikamy przycisk *OK* i wracamy do widoku *Projekt*. Gdy zaznaczymy utworzony blok opcji, możemy w panelu *Właściwości*, znajdującym się w dolnym prawym rogu okna, zmienić wygląd bloku (sekcje *Font* i *Wygląd* — rysunek A.11).



Rysunek A.11. Panel Właściwości

Po wykonaniu tych czynności można przetestować działanie aplikacji, naciskając klawisz *F5*. Jeżeli w kodzie nie było błędów, powinien on zostać przekompilowany, a w przeglądarce internetowej powinna się wyświetlić grupa opcji wyboru z podanymi w kodzie wartościami (rysunek A.12).



Rysunek A.12. Grupa opcji wyboru widoczna w przeglądarce

Kontrolki serwerowe

Tworząc projekt aplikacji internetowej, mamy do dyspozycji dwa rodzaje kontrolek serwerowych:

- *Kontrolki serwerowe HTML* — wyglądają jak zwyczajne znaczniki HTML, posiadają jednak dodatkowy atrybut `runat="server"`.
- *Kontrolki serwerowe ASP.NET* (ang. *ASP.NET server controls*) — nazywane również kontrolkami internetowymi, udostępniają spójny model obiektowy oraz spójne nazewnictwo właściwości. Opracowano je po to, by udostępnić wygodniejszy interfejs API do pracy ze standardowymi kontrolkami HTML. W przypadku kontrolek HTML istnieje wiele sposobów obsługi wprowadzania danych, np.:

```
<input type="radio">

<input type="checkbox">

<input type="button">

<input type="text">

<textarea>
```

Każda z podanych kontrolek działa inaczej i wymaga zastosowania innych atrybutów. Kontrolki serwerowe normalizują zbiory dostępnych elementów sterujących oraz zapewniają spójne wykorzystanie atrybutów w ich modelu obiektowym.

Na przykład podane wyżej kontrolki HTML jako kontrolki serwerowe będą miały postać:

```
<asp:RadioButton>
<asp:CheckBox>
<asp:Button>
```

Kod HTML przekazywany do przeglądarki nie zawiera znaczników serwerowych. ASP.NET konwertuje je na standardowy kod HTML. Kod wykonywany na serwerze może być dostarczany w postaci standardowego kodu HTML lub w postaci zgodnej z konwencjami używanymi we wszystkich klasach biblioteki .NET. To my decydujemy, jaka konwencja zostanie zastosowana.

Zadanie A.2

Utwórz nowy projekt pod nazwą *Test*. Wykorzystując możliwości definiowania różnych elementów formularza, zaprojektuj test dotyczący budowy komputera. Test powinien zawierać 10 pytań i dla każdego z nich trzy możliwości odpowiedzi utworzone za pomocą grupy opcji wyboru. Zapisz projekt pod nazwą *Test_komp*.

A.5.4. Powiązanie z bazą danych

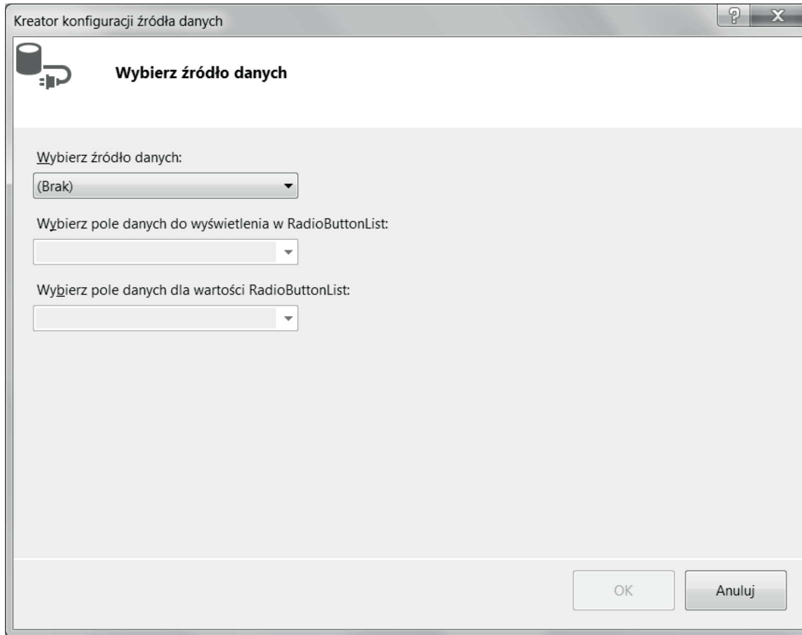
Na każdej stronie internetowej mogą pojawić się kontrolki wyświetlające dane, które w miarę upływającego czasu będą ulegały zmianom. Dane, które mogą ulegać zmianom, najlepiej przechowywać w bazach danych. W technologii ASP.NET kontrolki mogą zostać powiązane z danymi z bazy, co ułatwia prezentowanie tych danych i ich modyfikację.

W przykładzie A.24 w formularzu zostały umieszczone na stałe trzy opcje pozwalające użytkownikowi wybrać poziom wykształcenia. Możemy przyjąć, że z czasem dojdą nowe, np. licencjat, lub zostaną zmodyfikowane istniejące. Każda taka zmiana będzie prowadziła do zmian w kodzie aplikacji. Lepszym rozwiązaniem byłoby umieszczenie tych danych w bazie i powiązanie ich z przyciskami opcji wyświetlanymi w formularzu.

Przykład A.25

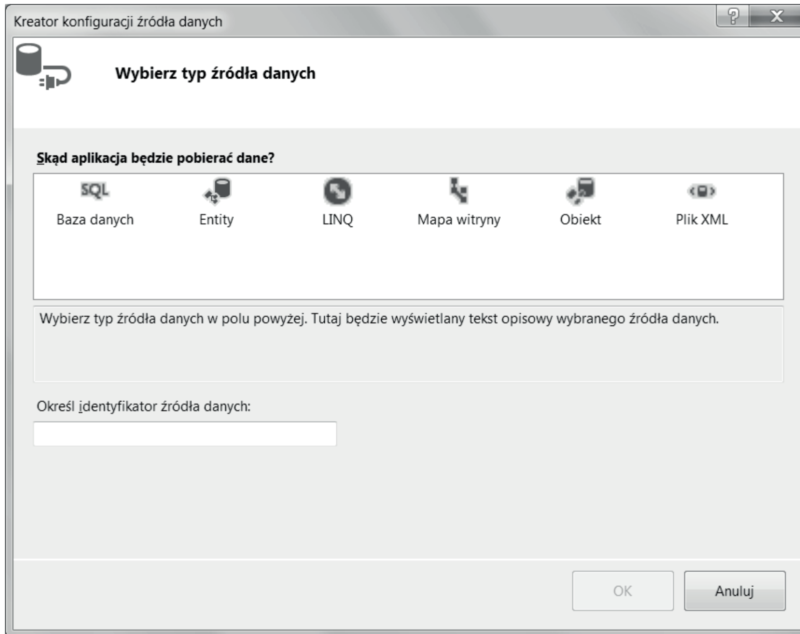
W tym przykładzie w formularzu też pojawi się grupa opcji, ale dane dla niej będą pobierane z bazy danych. Przed przystąpieniem do tworzenia formularza powinniśmy przygotować bazę z danymi. Uruchamiamy SQL Server i po połączeniu tworzymy nową bazę, a w niej tabelę `Wykształcenie`, która będzie zawierała dwa pola: `id_wyksztalcenia` i `wyksztalcenie`. Wprowadzamy do tabeli przykładowe dane, np. w polu `wyksztalcenie` wpisujemy: `podstawowe`, `średnie`, `wyższe`. Zamykamy bazę i kończymy pracę ze SQL Server. Wracamy do tworzenia aplikacji w Visual Studio. Do tworzonej aplikacji dodajemy nowy formularz *wyksztalcenie.aspx*. Umieszczamy w formularzu, podobnie jak w poprzednim, kontrolkę `RadioButtonList` pobraną

z *Przybornika*. W widoku *Projekt* zaznaczamy utworzony element. Przy zaznaczeniu elementu pojawi się inteligentny znacznik umożliwiający określenie źródła danych. Tym razem klikamy w nim opcję *Wybierz źródło danych*. Zostanie uruchomiony *Kreator konfiguracji źródła danych* (rysunek A.13).



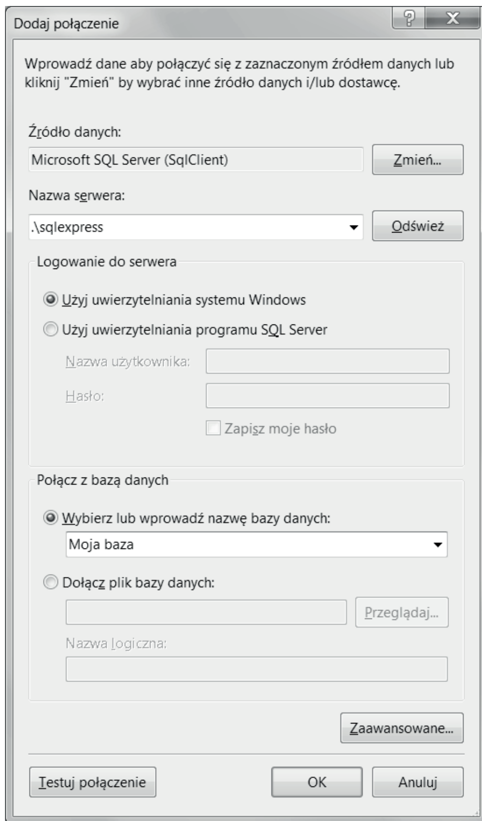
Rysunek A.13. Kreator konfiguracji źródła danych

Rozwijamy listę *Wybierz źródło danych* i wybieramy opcję *Nowe źródło danych*. Otworzy się kolejne okno (rysunek A.14), w którym należy wybrać typ źródła danych.



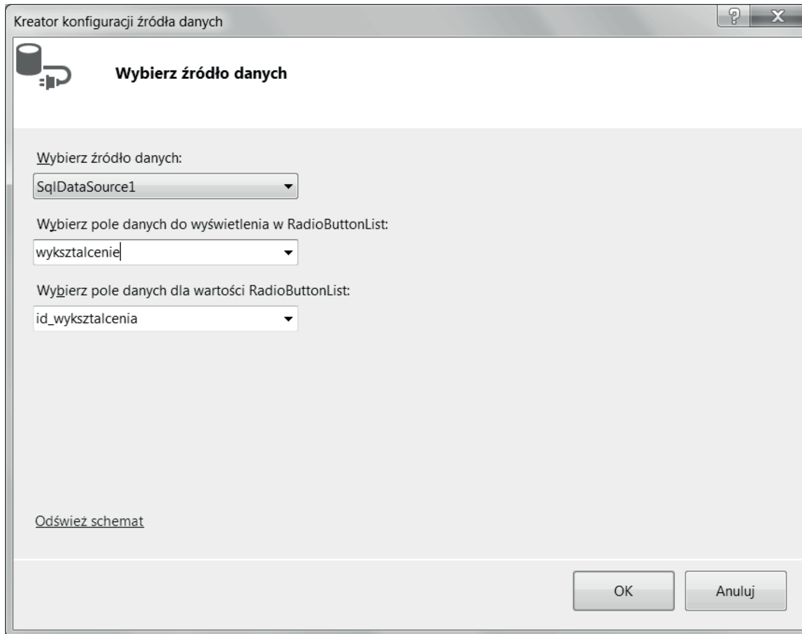
Rysunek A.14. Wybór źródła danych

Na liście zaznaczamy *Baza danych* i określamy identyfikator źródła danych (można zostawić zaproponowany przez system). Po kliknięciu przycisku *OK* wyświetli się okno *Wybierz połączenie danych*, w którym możemy wybrać istniejące połączenie z bazą danych lub utworzyć nowe. Tworzymy nowe połączenie — klikamy przycisk *Nowe połączenie…*. W polu *Źródło danych* wybieramy *Microsoft SQL Server* i w otwartym oknie *Dodaj połączenie* wypełniamy pola jak na rysunku A.15. Klikamy przycisk *Testuj połączenie*, aby sprawdzić połączenie z bazą danych.



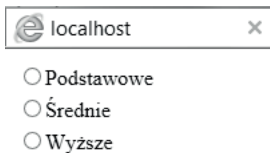
Rysunek A.15. Konfiguracja połączenia z bazą danych

Jeśli połączenie działa prawidłowo, klikamy przycisk *OK*. Właściwości połączenia zostaną zapisane w *Konfiguracji źródła danych*. W następnym oknie można zdecydować, czy parametry połączenia powinny zostać zapisane w pliku konfiguracji aplikacji. Jeżeli tak, to należy podać nazwę tego pliku. W kolejnym oknie wybieramy tabelę, która będzie przechowywała informacje potrzebne do wyświetlenia w przyciskach opcji, oraz kolumnę, z której będą pobierane dane. Powinny zostać zaznaczone pola `id_wyksztalcenia` oraz `wyksztalcenie`. Po kliknięciu przycisku *Następny* ponownie trzeba przetestować połączenie (przycisk *Zapytanie testowe*), aby sprawdzić, czy z tabeli są pobierane prawidłowe dane. Po zakończeniu ustalania źródła danych musimy powiązać utworzone źródło danych z kontrolką `RadioButtonList`. Kontrolka rozróżnia wartość wyświetlaną (pole `wyksztalcenie`) oraz wartość wyboru (pole `id_wyksztalcenia`). Pola te należy prawidłowo przypisać w kolejnym oknie kreatora (rysunek A.16).



Rysunek A.16. Przypisanie pól wyksztalcenie i id_wyksztalcenia

W celu sprawdzenia działania aplikacji naciskamy klawisz *F5*. Aplikacja zostanie skompilowana, a strona wyświetli się w przeglądarce (rysunek A.17).



Rysunek A.17. Wynik powiązania kontrolki RadioButtonList z bazą danych

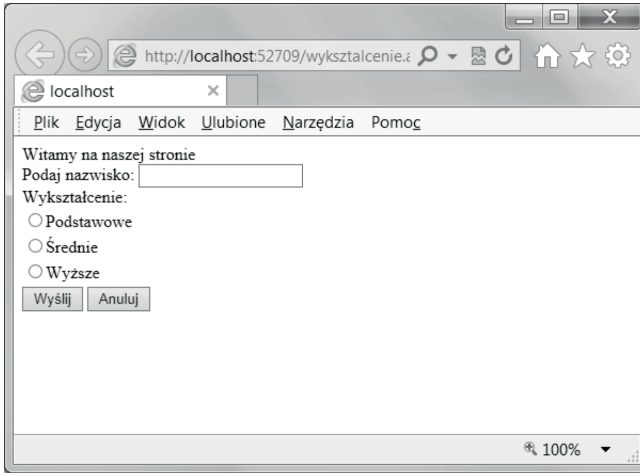
Aby utworzyć pełniejszy formularz, do tworzonej strony internetowej dodamy pole tekstowe, w którym użytkownik wprowadzi swoje nazwisko, oraz przyciski *Wyślij* i *Anuluj*. Działanie przycisku *Wyślij* będzie polegało na tym, że gdy użytkownik wprowadzi dane i kliknie ten przycisk, z pola tekstowego zostanie odczytane wprowadzone nazwisko, a z pola opcji zaznaczony poziom wykształcenia. Następnie wyświetli się komunikat informujący, jakie dane zostały wprowadzone.

Przechodzimy do widoku źródła kodu (*Źródło*) i powyżej przycisków opcji wyboru wstawiamy kontrolkę `TextBox`, a poniżej opcji wyboru dwie kontrolki `Button`. Do kontrolki `TextBox` dodajemy tekst `Podaj nazwisko:`, a dla kontrolek `Button` zmieniamy parametr `Text` na `Wyślij` i `Anuluj`. Po tych zmianach kod aplikacji będzie wyglądał jak w przykładzie niżej.

Przykład A.26

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="wyksztalcenie.
aspx.cs" Inherits="WebApplication6.wyksztalcenie" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>Witamy na naszej stronie</div>
    <div>
        Podaj nazwisko:
        <asp:TextBox ID="Nazw" runat="server"></asp:TextBox></div>
        <div>Wykształcenie:</div>
        <div>
            <asp:RadioButtonList ID="rblWyksztalcenie" runat="server"
DataSourceID="SqlDataSource1" DataTextField="wyksztalcenie"
DataValueField="id_wyksztalcenia"></asp:RadioButtonList>
            <asp:SqlDataSource ID="SqlDataSource1" runat="server" Con-
nectionString="<%$ ConnectionStrings:Do_ASPConnectionString2 %>"
SelectCommand="SELECT [id_wyksztalcenia], [wyksztalcenie] FROM
[Wyksztalcenie]"></asp:SqlDataSource>
        </div>
    <div>
      <asp:Button ID="Button1" runat="server" Text="Wyślij" OnClick="Button1_
Click" />
        <asp:Button ID="Button2" runat="server" Text="Anuluj" />
    </div>
    <div>
    <asp:Label id="lblMsg" runat="server"></asp:Label>
    </div>
    </form>
</body>
</html>
```

Po naciśnięciu klawisza *F5* strona wyświetlona w przeglądarce internetowej powinna wyglądać jak na rysunku A.18.



Rysunek A.18. Wynik wykonania kodu z przykładu A.26

Kolejnym etapem pracy z kodem będzie dodanie obsługi przycisku *Wyślij*. Kiedy użytkownik wprowadzi dane i kliknie przycisk *Wyślij*, powinny zostać odczytane wartość pola `nazwisko` oraz wartość opcji wyboru, a następnie powinien zostać wysłany komunikat potwierdzający wprowadzone dane.

Aby dodać obsługę przycisku *Wyślij*, przechodzimy do widoku *Projekt*, zaznaczamy przycisk *Wyślij* i dwukrotnie klikamy go myszą. Wyświetli się plik kodu ukrytego oraz zostanie utworzona procedura obsługi zdarzenia `Click` dla wybranego przycisku. Kod ten ma postać:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
namespace WebApplication6
{
    public partial class WebForm2 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
```

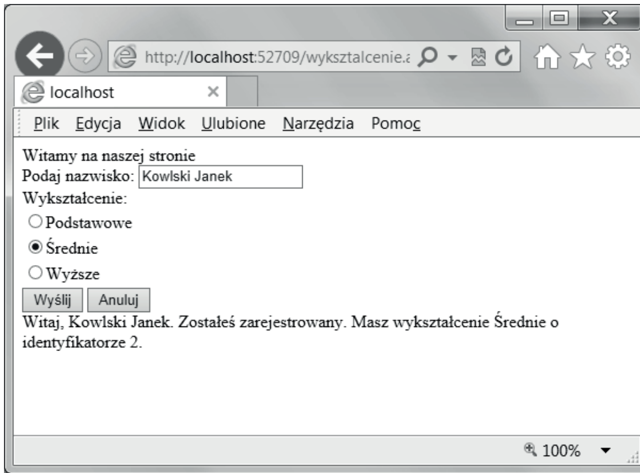
```
        }
        protected void Button1_Click(object sender, EventArgs e)
        {

        }
    }
}
```

W kodzie obsługującym zdarzenie umieścimy treść komunikatu, która się wyświetli, gdy nastąpi kliknięcie przycisku *Wyślij*. W komunikacie zostaną umieszczone dane odczytane z pola zawierającego nazwisko oraz z opcji wyboru wykształcenia. Zmodyfikowany plik kodu ukrytego może mieć postać:

```
namespace WebApplication6
{
    public partial class wyksztalcenie : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
        }
        protected void Button1_Click(object sender, EventArgs e)
        {
             lblMsg.Text = "Witaj, " + Nazw.Text.Trim() + ". Zostałeś za-
rejestrowany. " +
"Masz wykształcenie  " + rblWyksztalcenie.SelectedItem.Text + " o identy-
fikatorze " + rblWyksztalcenie.SelectedValue + ".";
        }
    }
}
```

Po uruchomieniu aplikacji i wprowadzeniu przez użytkownika danych oraz kliknięciu przycisku *Wyślij* efekt jej działania będzie taki jak na rysunku A.19.



Rysunek A.19. Wynik wykonania kodu po wprowadzeniu obsługi zdarzenia Click

Analiza kodu

Przeanalizujmy kod aplikacji, który został wygenerowany automatycznie.

Kluczowym elementem podczas tworzenia aplikacji jest element `Page`. Obiekt ten reprezentuje stronę, a dodatkowo zawiera grupę zdarzeń do innych kontrolek. Wyposażony jest również w kilka metod, które są istotne dla działania aplikacji. Należy do nich metoda `Page_Load()`, która wykonywana jest podczas wczytywania strony. Można w tej metodzie zdefiniować zdarzenia, które będą istotne, na przykład gdy strona będzie wczytywana w wyniku działania użytkownika. Dla elementu `Page` dostępna jest właściwość `IsPostBack`, która określa, czy strona jest wczytywana po raz pierwszy.

Przykład A.27

```
protected void Page_Load(object sender, EventArgs e)
  {
      if (!Page.IsPostBack)
      {
          TextBox1.Text = "";
      }
  }
```

Przykład pokazuje przygotowanie pustego pola tekstowego, gdy metoda uruchamiana jest po raz pierwszy.

Wszystkie zdarzenia kontrolek obsługiwane są za pomocą wbudowanej klasy `System.Web.UI.Controls`, która zawiera standardowe zdarzenia kierujące obsługą danej kontrolki. W naszym przykładzie po kliknięciu przez użytkownika przycisku *Wyślij* wystąpi zdarzenie `OnClick`. Jest ono powiązane z metodą `OnClick` i dla przycisku

Wyślij ma nazwę `Button1_Click`. Automatycznie zostanie uruchomiona metoda `Button1_Click`.

Metody uruchamiane przez komponenty platformy .NET są bez określonego typu (deklarowane są jako `void`) i mają dwa parametry. Pierwszy z nich ma nazwę `sender` i określa obiekt, który spowodował wywołanie danego zdarzenia, czyli np. przycisk, który został kliknięty. Drugi z parametrów jest obiektem, który przechowuje argumenty z dodatkową informacją dla systemu.

W naszym przykładzie w pliku **.aspx* został zdefiniowany przycisk, po którego kliknięciu będzie wywołana metoda `Button1_Click`:

```
<asp:Button ID="Button1" runat="server" Text="Wyślij" OnClick="Button1_
Click" />
```

W kodzie aplikacji została zdefiniowana jego obsługa, gdzie od razu widać przekazywane parametry:

```
protected void Button1_Click(object sender, EventArgs e)
        {
             lblMsg.Text = "Witaj, " + Nazw.Text.Trim() + ". Zostałeś za-
rejestrowany. " +
"Masz wykształcenie  " + rblWyksztalcenie.SelectedItem.Text + " o identy-
fikatorze " + rblWyksztalcenie.SelectedValue + ".";
        }
```

Zadanie A.3

Zmodyfikuj formularz utworzony w zadaniu A.2. Dane do testu powinny być pobierane z bazy danych. W tym celu utwórz bazę danych, która będzie zawierała pytania oraz trzy wersje odpowiedzi dla każdego pytania. Zastanów się, jak zaprojektować taką bazę. Następnie zdefiniuj połączenie z bazą danych i przetestuj działanie formularza.

A.5.5. Przemieszczanie się między stronami

Przemieszczanie się między stronami aplikacji może być realizowane na kilka sposobów. Najprostszym z nich jest zastosowanie mechanizmu hiperłącza. Obiektem, który realizuje to zadanie, jest kontrolka `Hyperlink`. Posiada ona właściwość `NavigateURL` przechowującą adres strony, do której ma nastąpić przekierowanie. Adres może prowadzić do strony platformy ASP.NET (*strona.aspx*) lub stron tworzonych w innych technologiach (*strona.php*, *strona.html*).

Przykład A.28

Tworzymy nowy projekt. W utworzonym projekcie wybieramy z menu *Projekt/Dodaj nowy element…*, a z listy szablonów *Formularz sieci Web*. Formularz nazwiemy `Form1`. Umieszczamy w nim tekst *Jesteś na stronie głównej*. Dodajemy do aplikacji kolejny

formularz, `Form2` (w menu wybieramy *Projekt/Dodaj nowy element…*). Umieszczamy w nim tekst *Jesteś na stronie drugiej*. Dodajemy trzeci formularz, `Form3`, i umieszczamy w nim tekst *Jesteś na stronie trzeciej*. Wracamy do formularza `Form1` i dodajemy z *Przybornika* kontrolkę `Hyperlink`. W widoku *Projekt* klikamy utworzoną kontrolkę i w panelu *Właściwości* odnajdujemy właściwość `NavigateURL`. Klikamy ikonę znajdującą się w polu i w otwartym oknie, w polu *Zawartość folderu*, wybieramy stronę, do której powinno przenieść nas hiperłącze. W naszym przypadku będzie to plik *WebForm2.aspx*. Zatwierdzamy wprowadzone zmiany. Podobnie postępujemy z kontrolką `Hyperlink` dla pozostałych formularzy. Po uruchomieniu aplikacji w przeglądarce można sprawdzić, jak funkcjonują zdefiniowane hiperłącza.

A.5.6. Pobieranie danych z bazy

Wiele aplikacji internetowych wykorzystuje dane zewnętrzne pochodzące z bazy np. SQL. Dostęp do tak przechowywanych danych może być realizowany na różne sposoby.

Przykład A.29

Utworzymy prostą aplikację, która będzie pobierała dane z bazy SQL Server. Będziemy pobierali z bazy danych *Ksiegarnia_internetowa* (opisanej w części II podręcznika) dane na temat książek dostępnych w księgarni. Zakładamy, że aplikacja ma uprawnienia potrzebne do uzyskania dostępu do danych. Tworzymy nowy projekt, a w nim tworzymy nowy element, wybierając z listy szablonów *Formularz sieci Web*. Następnym krokiem będzie zdefiniowanie połączenia z bazą danych. Wybieramy z menu *Narzędzia/Łączenie z bazą danych…* i w otwartym oknie konfigurujemy połączenie. W polu *Źródło danych* zostawiamy *Microsoft SQL Server (SqlClient)*, w polu *Nazwa serwera* wpisujemy nazwę instancji serwera bazodanowego, np. `.\sqlexpress`. W polu *Logowanie do serwera* zostawiamy domyślny sposób uwierzytelnienia systemu Windows i wprowadzamy nazwę bazy danych, w naszym przypadku *Ksiegarnia_internetowa*. Testujemy połączenie i jeżeli połączenie zostało nawiązane, kończymy jego konfigurowanie. Następnym krokiem będzie wybór danych, które będą wyświetlane na stronie. Najprostszym sposobem jest przeciągnięcie wybranej tabeli na formularz. W tym celu wybieramy dla tworzonego formularza widok *Projekt*. W panelu *Eksplorator bazy danych* (powinien znajdować się z prawej strony okna) rozwijamy opcję z nazwą bazy danych (*Ksiegarnia_internetowa*) i wybieramy w pozycji *Tabele/Ksiazki*. Chwytamy zaznaczoną tabelę i przeciągamy na formatkę formularza. W oknie *Projekt* pojawią się wstawiona tabela oraz obiekt typu `SqlDataSource`. Widoczna tabela jest obiektem klasy `GridView`. Po jej zaznaczeniu pojawi się *inteligentny znacznik*, który umożliwia określenie parametrów wstawionego obiektu. Uzyskany efekt można obejrzeć w przeglądarce, naciskając na klawiaturze *F5*. Na stronie powinna się wyświetlić zawartość tabeli `Ksiazki`.

Możemy wrócić do utworzonego kodu (widok *Źródło*). Widać, że zostały zdefiniowane dwa wspomniane wyżej obiekty, `SqlDataSource1` i `GridView1`. W kodzie występują też jawnie zdefiniowane nazwy oraz właściwości kolumn tabeli `Ksiazki`. Obiektem, który dostarcza danych do wyświetlania, jest `SqlDataSource1`. Jego właściwości

`SelectCommand`, `UpdateCommand`, `InsertCommand` i `DeleteCommand` zawierają kod języka SQL wykorzystywany przez obiekty klasy `SqlDataSource`. Natomiast `ProviderName` definiuje parametry połączenia z bazą danych.

W widoku `Projekt` zaznaczamy widoczną tabelę. Klikamy *inteligentny znacznik* i wybieramy opcję *Edytuj kolumny...*. W otwartym oknie w obszarze *Wybrane pola* możemy określić, które pola tabeli powinny się wyświetlić na stronie. Po wybraniu opcji *Autoformatowanie* możemy wybrać schemat kolorystyczny wyświetlanej tabeli. Tabela wyświetlona w przeglądarce może mieć postać podobną do tej z rysunku A.20.

localhost

id_ksiazki	tytul	cena	wydawnictwo	temat	jezyk_ksiazki
3	Antygona	20,0000	Literatura	dramat	polski
4	Balladyna	10,0000	Nowa	dramat	polski
5	Chłopi	25,0000	Nowa	powieść	polski
6	Dziady	32,0000	Odys	dramat	polski
7	Faraon	34,0000	Nowa	powieść	polski
8	Fraszki	14,0000	Literatura	fraszki	polski
9	Grażyna	15,0000	Nowa	poemat	polski
10	Kamizelka	10,0000	Literatura	nowela	polski
11	Katarynka	9,0000	Literatura	nowela	polski
12	Kazania sejmowe	14,0000	Nowa	proza	polski
13	Konrad Wallenrod	16,0000	Odys	poemat	polski
14	Kordian	21,0000	Nowa	dramat	polski
15	Krzyżacy	37,0000	Nowa	powieść	polski
16	Lalka	39,0000	Literatura	powieść	polski

Rysunek A.20. Wyświetlanie na stronie danych pobranych z bazy SQL

Oprócz możliwości zmiany wyglądu utworzonego obiektu mamy możliwość zmiany jego funkcjonalności. Na przykład możemy ustawić możliwość zmiany z poziomu przeglądarki zawartości wyświetlanej tabeli. Wprowadzone z poziomu przeglądarki zmiany w danych zapisanych do tabeli będą widoczne w bazie SQL.

Przykład A.30

W celu dodania nowej funkcjonalności musimy ponownie skonfigurować źródło danych. W widoku *Projekt* zaznaczamy obiekt `SqlDataSource1`, klikamy strzałkę z prawej strony i wybieramy opcję *Konfiguruj źródło danych...*. Przechodzimy do okna *Konfiguruj instrukcję Select*, klikamy przycisk *Zaawansowane...* i w otwartym oknie zaznaczamy opcję *Generuj instrukcje INSERT, UPDATE, DELETE*. Kończymy konfigurację połączenia.

W widoku *Projekt* zaznaczamy tabelę i w panelu właściwości ustawiamy następujące właściwości:

```
AutoGenarateDeleteButton = true

AutoGenarateEditButton = true
```

Po otwarciu strony w przeglądarce zobaczymy w tabeli nową kolumnę zawierającą opcje *Edytuj* i *Usuń*, za pomocą których można edytować i usuwać dane w tabeli (rysunek A.21).

localhost ×

	id_ksiazki	tytul	cena	wydawnictwo	temat	jezyk_ksiazki
Aktualizuj Anuluj	3	Antygona	20,0000	Literatura	dramat	polski
Edytuj Usuń	4	Balladyna	10,0000	Nowa	dramat	polski
Edytuj Usuń	5	Chłopi	25,0000	Nowa	powieść	polski
Edytuj Usuń	6	Dziady	32,0000	Odys	dramat	polski
Edytuj Usuń	7	Faraon	34,0000	Nowa	powieść	polski
Edytuj Usuń	8	Fraszki	14,0000	Literatura	fraszki	polski
Edytuj Usuń	9	Grażyna	15,0000	Nowa	poemat	polski
Edytuj Usuń	10	Kamizelka	10,0000	Literatura	nowela	polski

Rysunek A.21. Możliwość edytowania na stronie danych pobranych z bazy SQL

W panelu *Przybornik*, w obszarze *Dane*, znajdują się kontrolki, które mogą zostać wykorzystane do obsługi bazy danych. Jedną z nich, `DetailsView`, możemy wykorzystać do dodawania nowych wpisów do tabeli. Przeciągamy tę kontrolkę na formularz. Klikamy strzałkę z jej prawej strony (inteligentny znacznik) i jako źródło danych wskazujemy obiekt `SqlDataSource1` oraz zaznaczamy opcję *Włącz wstawianie*. Gdy wyświetlimy utworzoną aplikację, w przeglądarce oprócz tabeli z danymi będzie widoczny formularz do wprowadzania nowych danych do bazy (rysunek A.22). Wprowadzone przez formularz dane zostaną zapisane w bazie danych SQL.

Edytuj Usuń	19	Ogniem i mieczem	39,0000	Literatura	powieść
Edytuj Usuń	20	Oda do młodości	21,0000	Odys	wiersze
Edytuj Usuń	25	Ulisses	46,0000	Literatura	powieść

tytul	
Id_autora	
cena	
wydawnictwo	
temat	
miejsce_wydania	
jezyk_ksiazki	
opis	
Wstaw Anuluj	

Rysunek A.22. Wyświetlany na stronie formularz do wprowadzania nowych danych do bazy

Zadanie A.4

Utwórz formularz, w którym będą wyświetlane dane dotyczące klientów księgarni internetowej. Umieść w nim kontrolkę umożliwiającą wprowadzanie danych nowych klientów. Powiąż utworzony formularz z bazą danych *Ksiegarnia_internetowa* i wykorzystaj tabelę `Klient`.

A.5.7. Obsługa wyjątków

Z różnych powodów aplikacja może działać nieprawidłowo. Powodem może być celowe działanie użytkownika, niepoprawnie skonfigurowany serwer lub też niezgodne z oczekiwaniami zachowanie aplikacji w pewnych sytuacjach. Obsługa błędów jest jednym

z istotniejszych elementów tworzonych aplikacji. W języku C# istnieje możliwość obsługiwania wyjątków, czyli nieprzewidzianych w programie sytuacji wyjątkowych. Służy do tego konstrukcja:

```
try {
//kod potencjalnie niebezpieczny }
catch  {
//przechwycenie wyjątku }
```

Bloków `catch` w instrukcji może być kilka, a każdy z nich może przechowywać określony wyjątek. Istotne jest, aby był jeden blok `catch`, który będzie przechwytywał wszystkie wyjątki niewymienione we wcześniejszych blokach `catch`. Gdy wystąpi wyjątek, program porówna go z wyjątkami zawartymi w blokach `catch` w kolejności, w jakiej zostały zapisane, czyli wyjątek obsługujący wszystko powinien znaleźć się na końcu listy wyjątków.

Przykład A.31

Zakładamy, że utworzona aplikacja będzie wyświetlała dane pobrane z pliku tekstowego. Do otwarcia pliku zostanie użyta metoda `Page_Load()` w postaci:

```
protected void Page_Load(object sender, EventArgs e)
 {
        System.IO.FileStream fs = System.IO.File.Open(@"c:\tekst.txt",
System.IO.FileMode.Open);
 }
```

Jeżeli podana ścieżka dostępu do pliku będzie nieprawidłowa, zamiast jego zawartości w przeglądarce pojawi się komunikat o błędzie serwera (rysunek A.23).

Błąd serwera w aplikacji '/'.

Nie można odnaleźć pliku 'c:\tekst.txt'.

Opis: Podczas wykonywania bieżącego żądania sieci Web wystąpił nieobsługiwany wyjątek. Aby uzyskać dodatkowe informacje o błędzie i miejscu jego występowania w kodzie, przejrzyj ślad stosu.

Szczegóły wyjątku: System.IO.FileNotFoundException: Nie można odnaleźć pliku 'c:\tekst.txt'.

Błąd źródła:

```
Wiersz 12:        protected void Page_Load(object sender, EventArgs e)
Wiersz 13:        {
Wiersz 14:            System.IO.FileStream fs = System.IO.File.Open(@"c:\tekst.txt", System.IO.FileMode.Open);
Wiersz 15:        }
Wiersz 16:    }
```

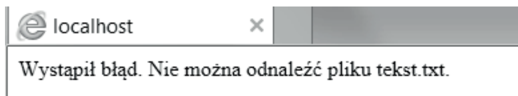
Rysunek A.23. Komunikat o błędzie aplikacji wyświetlony przez przeglądarkę

Tego typu informacje nie powinny pojawiać się na stronie, ponieważ użytkownikom strony będzie to przeszkadzało, a innym osobom może posłużyć do zdobycia informacji na temat środowiska dostępnego na serwerze.

Po dodaniu obsługi wyjątków zmodyfikowany kod będzie wyglądał następująco:

```
protected void Page_Load(object sender, EventArgs e)
{
    try
   {
    System.IO.FileStream fs = System.IO.File.Open(@"c:\tekst.txt", System.
IO.FileMode.Open);
}
   catch
   {
     Response.Write("Wystąpił błąd. Nie można odnaleźć pliku tekst.txt.");
   }
}
```

W podanym przykładzie następuje próba otwarcia pliku *tekst.txt*. Plik nie istnieje, więc wykonanie polecenia jest niemożliwe i dlatego rozpoczyna się wykonywanie kodu obsługującego wyjątek (rysunek A.24).



Rysunek A.24. Obsługa wyjątków

Dla tworzonej aplikacji można zdefiniować globalną obsługę błędów. W tym celu należy dodać do tworzonego projektu plik *Web.config*. Dodajemy taki plik, wybierając z menu *Plik/Otwórz plik…/* i w otwartym oknie wskazując plik *Web*. Następnie w otwartym pliku, w bloku między znacznikami `<system.web>` i `</system.web>`, wstawiamy kod:

```
<customErrors mode="On" defaultRedirect="GenericErrorPage.htm">
 <error statusCode="403" redirect="NoAccess.htm"/>
 <error statusCode="404" redirect="FileNotFound.htm"/>
 </customErrors>
```

Do pliku konfiguracyjnego zostanie dodana obsługa standardowych błędów `403` i `404`. W przypadku wystąpienia tych błędów nastąpi przekierowanie na zdefiniowane w pliku strony. Domyślną stroną, na którą będzie następowało przekierowanie w przypadku wystąpienia innych błędów, jest strona *GenericErrorPage.htm*.

Kody niektórych standardowych błędów zostały podane w tabeli A.5.

Tabela A.5. Kody niektórych standardowych błędów

Kod	Opis
401	Użytkownik nie ma odpowiednich uprawnień do oglądania zawartości strony
403	Serwer odmawia wykonania żądania
404	Szukana strona nie została znaleziona na serwerze
408	Przekroczony został czas oczekiwania na odpowiedź serwera
500	Wewnętrzny błąd serwera. Kod ten reprezentuje wszystkie nieobsługiwane wyjątki w aplikacjach działających na serwerze
505	Wersja protokołu HTTP używana przez klienta jest nieobsługiwana przez serwer

Można też definiować obsługę błędów na poziomie poszczególnych plików **.aspx*. W takim przypadku należy w dyrektywie `<%@Page...%>`, która znajduje się na początku strony, dodać właściwość `errorPage` w postaci:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" ErrorPage ="~/ErrDefault.aspx" %>
```

Gdy wystąpi nieobsługiwany wyjątek, nastąpi przekierowanie na stronę *ErrDefault.aspx*.

PYTANIA KONTROLNE

1. Wymień podstawowe cechy języka obiektowego.
2. Jaką rolę na platformie .NET odgrywają przestrzenie nazw?
3. Kiedy w języku C# stosujemy metodę `Main`?
4. Jaka zależność w języku C# występuje między klasami i obiektami?
5. Co oznacza określenie „przeciążenie funkcji” w języku C#?

ZADANIA

1. W MS SQL Server utwórz bazę danych *Moja_szkoła*. Baza danych powinna zawierać tabelę `Uczeń` z danymi uczniów.
2. Utwórz formularz, w którym będą wyświetlane dane uczniów. Umieść w nim kontrolkę umożliwiającą wprowadzanie danych nowych uczniów. Powiąż utworzony formularz z bazą danych *Moja_szkoła* tak, aby dane nowych uczniów były zapisywane w tabeli `Uczeń`.