

Platforma .NET i język C#

Dariusz Mikułowski

Instytut Informatyki Uniwersytetu Przyrodniczo-Humanistycznego W Siedlcach

20 grudnia 2020

1 Co to jest .NET

.NET to rozbudowana platforma do tworzenia złożonych aplikacji. Od ok 2000 r. jest rozwijana przez firmę Microsoft, dlatego początkowo była częścią systemu operacyjnego Windows. Obecnie jest zestawem narzędzi dla wielu systemów operacyjnych. Można w niej tworzyć oprogramowanie dla Android, iOS, Linux, macOS, Windows oraz w Internecie i chmurze.

2 Składniki platformy .NET

Platforma .NET składa się z kilku głównych elementów:

- ★ środowiska .NET Framework,
- ★ narzędzi dla programistów,
- ★ serwerów .NET Enterprise.

3 Co jest potrzebne do programowania na platformie .NET?

Do programowania aplikacji na platformie .NET minimalnie wystarczy posiadać:

- ★ system Windows z zainstalowanym serwerem www np. IIS,
- ★ platformę .NET Framework
- ★ dowolny edytor tekstu

W praktyce, do tworzenia oprogramowania .NET wykorzystuje się jedno ze specjalistycznych środowisk programistycznych takich jak np.: standardowy produkt Microsoft czyli Visual Studio. Obecna jego najnowsza wersja to Visual Studio 2019. Środowisko oprócz darmowej wersji ultimate jest też dostępne w modelu subskrypcyjnym w różnych zestawach. od samego środowiska Visual Studio, poprzez platformę Azure, oprogramowanie do programowania i testowania, pomoc techniczną, szkolenia itp. W pracy nad aplikacjami .NET można wykorzystywać także inne środowiska takie jak: Visual Studio Web Developer Express Edition, Borland C Builder, Sharp-Developer, czy ASP.NET Web Matrix.

4 Składniki środowiska .NET Framework

Środowisko programistyczne .NET Framework to technologie, z których może korzystać programista przy tworzeniu aplikacji. W jego skład wchodzi:

- ★ Class Libraries (CL) - biblioteki klas. Zawierają one gotowe komponenty, funkcje i biblioteki, które można wykorzystać we własnym programie.
- ★ Common Language Runtime (CLR) - wspólne środowisko uruchomieniowe. Odpowiada ono za kompilowanie aplikacji .NET oraz ich prawidłowe działanie, a także za takie czynności jak przydzielanie i zwalnianie pamięci
- ★ Common Type System (CTS) - wspólny system typów. Dzięki niemu jest możliwa współpraca pomiędzy komponentami aplikacji, które były pisane w różnych językach.

5 Narzędzia dostępne w .Net

W roku 2005 została wydana druga wersja środowiska .Net Framework 2.0. Wraz z nią udostępniono darmowe, lżejsze od standardowego zintegrowane środowisko programistyczne Visual Studio 2005 Express, które składało się z kilku osobnych produktów:

- ★ Visual Basic 2005 Express Edition,
- ★ Visual C 2005 Express Edition,
- ★ Visual C++ 2005 Express Edition,
- ★ Visual J 2005 Express Edition,
- ★ Visual Web Developer 2005 Express Edition,
- ★ SQL Server 2005 Express Edition.

6 Wersje platformy i środowiska Visual Studio

Wersja platformy	Data wydania	Wersja Visual Studio	Dołączona do Windows
1.0	2002-02-13	Visual Studio .NET	Windows Server 2003, Windows XP
1.1	2003-04-24	Visual Studio .NET 2003	
2.0	2005-11-07	Visual Studio 2005	
3.0	2006-11-06		Windows Vista, Windows Server 2008

7 Wersje platformy i środowiska Visual Studio

Wersja platformy	Data wydania	Wersja Visual Studio	Dołączona do Windows
3.5	2007-11-19	Visual Studio 2008	Windows 7, Windows Server 2008 R2
4.0	2010-04-12	Visual Studio 2010	
4.5	2012-08-15	Visual Studio 2012	Windows 8

8 Wersje platformy i środowiska Visual Studio

Wersja platformy	Data wydania	Wersja Visual Studio	Dołączona do Windows
4.5.1	2013-10-12	Visual Studio 2013	Windows 8.1
4.6	2015-07-20	Visual Studio 2015	Windows 10
4.7	2017-04-05	Visual Studio 2017	Windows 10 v1703
4.7.2	2018-04-30	Visual Studio 2017	Windows 10 v1803
4.8	20-11-2019	Visual Studio 2019 ostatnia wersja	Windows 10

9 Czym jest .NET Core?

Jest to nowy framework, który był opracowywany równolegle do .NET 4.6 (Visual Studio 2015). Poprzednio nosił nazwę .NET 5.0. To nazewnictwo było powodem zamieszania wśród programistów ponieważ sugerowało, że jest to następca .NET 4.6, co nie jest prawdą. Jest to zupełnie nowy framework zbudowany od podstaw. Wraz z udostępnieniem nowego framework'a pojawiły się nowe jego składniki: CoreCLR oraz CoreFx. Zastąpiły one tradycyjny CLR oraz FCL (Framework Class Library), które były przypięte do platformy Windows.

CoreCLR zawiera w sobie m.in. kompilator oraz bazowe typy danych platformy .NET, a także wiele klas niskiego poziomu. Na samym szczycie nowego framework'a jest ASP.NET Core 1.0 oraz ASP.NET Core MVC.

10 Co nowego w .NET Core?

Wieloplatfromowość, - można tworzyć oprogramowanie pracując w Windows, Linux, Mac. W .NET framework trzeba było pracować w Visual Studio dostępnym tylko dla Windows. Teraz nie jest już ono wymagane. Można używać dowolnego edytora tekstowego a program może być wykonany przy użyciu konsoli.

Otwartość oprogramowania, Kod źródłowy .NET Core jest dostępny na rozproszonym systemie kontroli wersji - GIT.

Optymalizacja bibliotek i środowiska uruchomieniowego, .NET Core wspiera teraz nuget packages dla każdej składowej dużej biblioteki. Oznacza to, że deklarując przykładowo użycie biblioteki System.net w swoim programie nie jest za każdym razem ładowana cała biblioteka system tylko ten jej składnik, który akurat jest potrzebny.

11 Co nowego w .NET Core?

.NET Core zawiera aplikację konsolową zwaną dotnet.exe, która pozwala na utworzenie i wykonanie nowej aplikacji na dowolnej platformie.

Modularna budowa - wraz ze zmianami nie będą wydawane nowe wersje frameworka ale wszystkie nowe biblioteki będą dostępne jako nowe paczki nugget package.

Środowisko gotowe na chmurę, przy użyciu .NET Core możemy budować aplikacje dla tzw. sklepów z aplikacjami podłączonych do chmury takich jak: Web Apps czy IOT apps.

12 Języki programowania wspierane w .NET

Każdy język wspierany w .Net, musi spełniać specyfikację Common Language Infrastructure (CLI). Dzięki temu programując w każdym z tych języków programista może używać wszystkich składników biblioteki .Net.

Podstawowe języki dostarczane przez Microsoft:

- ★ C#
- ★ Visual Basic .NET
- ★ F#
- ★ C++/CLI (wcześniej Managed C++, wariant C++)
- ★ J# (wariant języka Java opracowany przez Microsoft)
- ★ JScript .NET (kompilowany wariant języka JScript)

13 Języki programowania wspierane w .NET

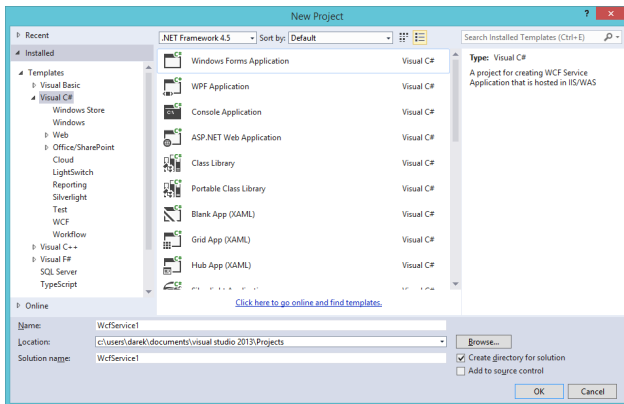
Pozostałe ważniejsze języki programowania:

- ★ COBOL
- ★ Delphi (Delphi.NET – od wersji 8 środowiska)
- ★ Eiffel
- ★ Fortran
- ★ Lisp
- ★ Nemerle (opracowany przez wrocławskich naukowców)
- ★ Perl
- ★ Python
- ★ Smalltalk

Głównymi językami dedykowanymi specjalnie dla platformy .NET są jednak C# i Visual Basic .NET.

Platforma .NET
.Net Core
Języki programowania wspierane w .NET
Prosta aplikacja konsolowa
Język C#
Konwersja typów w C#
Operacje na łańcuchach znakowych
Tablice
Struktury

14 Rodzaje aplikacji

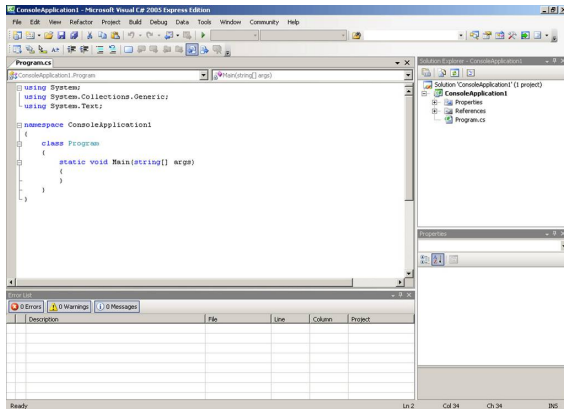


15 Programowanie wizualne

W środowisku Visual Studio wiele elementów można dodać do swojej aplikacji korzystając z tzw. programowania wizualnego. Polega ono na "składaniu" całej aplikacji z gotowych komponentów (klocków) które odbywa się na formularzu. Dotyczy to głównie elementów interfejsu użytkownika przyszłej aplikacji, ale nie tylko. Elementy aplikacji są wizualnie reprezentowanymi obiektami, które posiadają własności i metody. Środowisko programistyczne pomaga w składaniu całej aplikacji, określaniu właściwości i używaniu oraz implementowaniu metod.

Platforma .NET
.Net Core
Języki programowania wspierane w .NET
Prosta aplikacja konsolowa
Język C#
Konwersja typów w C#
Operacje na łańcuchach znakowych
Tablice
Struktury

16 Tworzenie aplikacji konsolowej



17 Przykład kodu prostego programu konsolowego

```
class witaj
{
    static void Main(string[] args)
    {
        System.Console.WriteLine("Witaj Świecie!");
    }
}
```

18 Metody

W C# programy budujemy z metod. Mamy bardzo dużo dostępnych metod w bibliotekach .Net Np. metoda `System.Console.WriteLine()`, która wypisuje tekst na konsoli. Jej parametrem jest łańcuch tekstowy. Obowiązkowym elementem każdej metody są nawiasy, w których podaje się parametry. Jednak nie wszystkie metody mają parametry. Metody bezparametrowe piszemy z pustymi nawiasami. np.: `System.Console.Read()`; . Każda metoda musi zwracać jakiś konkretny typ danych. Ale jeśli nie chcemy aby tak było możemy użyć typu pustego o nazwie `void` np.

```
Public Static void Main(String[] args){
```

19 Klasy

Podobnie jak w języku Java również w C# używamy klas.
.NET Framework udostępnia szereg klas gotowych do użycia.
Przykładowo do obliczenia pierwiastka z liczby możemy użyć klasy `System.Math` i jej metody `SQRT(...)`.

20 Wbudowane typy danych

W języku C# mamy do dyspozycji dużo wbudowanych typów danych takich jak np.:

- ★ byte - liczba od 0 do 255
- ★ sbyte - liczba od -128 do 127
- ★ short - liczba od -32768 do 32767
- ★ int - liczba od -2147483648 do 2147483647
- ★ uint - liczba od 0 do 4294967295
- ★ Long - liczba od -9223372036854775808 do 9223372036854775807
- ★ ulong - liczba od 0 do 18446744073709551615
- ★ float - liczba od -3,402823e38 do 3,402823e38
- ★ double - liczba od -1,79769313486232e308 do 1,79769313486232e308
- ★ decimal - liczba od -79228162514264337593543950335 do 79228162514264337593543950335
- ★ char - Pojedynczy znak
- ★ string - Łańcuch znaków typu char
- ★ bool - wartość logiczna true lub false

21 Stałe

W języku C# istnieją też stałe. Deklarujemy je podając ich typ i nazwę poprzedzając deklarację słowem kluczowym `const`. np.

```
const double Version = 1.0;
```

Jeśli w dalszym ciągu kodu napiszemy: `Version = 2.0;` kompilator wskaże błąd, gdyż nie można zmieniać wartości stałej. Stałych używamy po to aby nazwać liczbowe lub znakowe wartości, które łatwiej jest nam następnie użyć rozpoznając ich nazwę, a nie wartość.

22 Operacje na konsoli

Podstawowe operacje wykonywane na konsoli to wypisywanie znaków na ekran oraz ich odczyt z klawiatury. Do ich realizacji służą metody `WriteLine()` - wypisująca linię tekstu, `ReadLine()` - odczytująca wprowadzoną linię tekstu, a także `Write()` - wypisująca znak i `Read()` odczytująca jeden znak wprowadzony z klawiatury.

Przykładowy prosty program z użyciem tych metod:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Cześć, jak masz na imię?");
        string name; // deklaracja zmiennej
        name = Console.ReadLine();
        // pobranie tekstu wpisanego przez użytkownika
        Console.WriteLine("Miło mi " + name + ". Jak się masz?");
        Console.ReadLine();
    }
}
```

23 Metody klasy konsoli

Innymi najważniejszymi metodami konsoli są:

- ★ Beep(...) - Umożliwia odegranie dźwięku z głośnika systemowego.
- ★ Clear() - Czyści ekran konsoli.
- ★ ResetColor() - Ustawia domyślny kolor tła oraz tekstu.
- ★ SetCursorPosition(...) - Ustawia pozycję kursora w oknie konsoli.
- ★ SetWindowPosition(...) - Umożliwia ustawienie położenia okna konsoli.
- ★ SetWindowSize(...) - Umożliwia określenie rozmiaru okna konsoli.

24 Metody konsoli

Przykładowy program z użyciem metod konsoli:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        // ustawiamy rozmiar okna
        Console.SetWindowSize(60, 30);
        // ustawiamy położenie tekstu
        Console.SetCursorPosition(10, 10);
        Console.WriteLine("Witaj świecie!");
        Console.ReadLine();
        // czyścimy ekran
        Console.Clear();
        // odgrywamy dźwięk
        Console.Beep();
    }
}
```

25 Operatory

W języku C# tak jak w Java możemy używać wielu operatorów. Istnieją tu takie rodzaje operatorów jak:

- ★ Relacyjne
- ★ Arytmetyczne
- ★ Inkrementacji i dekrementacji
- ★ Logiczne
- ★ Bitowe,
- ★ przypisania

26 Operatory bitowe

Operatory bitowe oferują działania na liczbach binarnych. Umożliwiają wykonywanie operacji na bitach, a nie na liczbach. W niektórych sytuacjach ich użycie jest wygodniejsze od zastosowania operatorów logicznych lub arytmetycznych. Możemy używać następujących operatorów bitowych:

- ★ Koniunkcja &
- ★ Zaprzeczenie ~
- ★ Alternatywa |
- ★ Dysjunkcja ^
- ★ Przesunięcie w lewo <<
- ★ Przesunięcie w prawo >>
- ★ uzupełnienie bitowe ~

27 Inne operatory

Inne operatory, których możemy używać do różnych zastosowań:

- ★ (T)x - rzutowanie zmiennej x na typ T
- ★ new - powołanie nowej instancji obiektu lub nowej klasy
- ★ checked i unchecked - odpowiednio wymuszenie lub zabronienie sprawdzania przepełnienia podczas wykonywania operacji arytmetycznej,
- ★ is - sprawdzenie czy zmienna jest danego typu
- ★ as - konwersja typów podobnych
- ★ && - warunkowa koniunkcja
- ★ || warunkowa alternatywa
- ★ ?: warunek złożony (może zastąpić konstrukcję if else)

28 Instrukcje warunkowe

Podstawową instrukcją warunkową w języku C# jest instrukcja if.
Jej bardziej złożona odmiana to instrukcja if else.
Przykładowe zastosowanie instrukcji if z else:

```
string imie = Console.ReadLine();  
if (imie[imie.Length - 1] == 'a')  
{  
    if (DateTime.Now.Hour >= 18)  
    {  
        Console.WriteLine("Dobry wieczór koleżanko!");  
    } else {  
        Console.WriteLine("Dzień dobry koleżanko!");  
    } } else {  
    Console.WriteLine("Witaj kolego!");  
}  
}
```

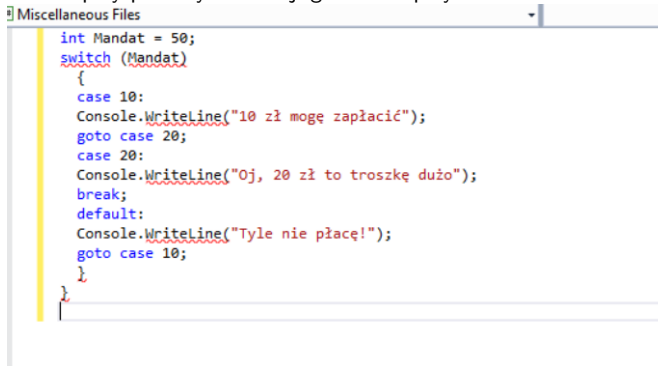
29 Instrukcja switch

Instrukcja switch umożliwia sprawdzanie wielu warunków jednocześnie. W C# używa się jej razem z instrukcjami case oraz break. Możemy też użyć instrukcji default. Np:

```
string imie = Console.ReadLine();  
if (imie[imie.Length - 1] == 'a')  
{  
    if (DateTime.Now.Hour >= 18)  
    {  
        Console.WriteLine("Dobry wieczór koleżanko!");  
    } else {  
        Console.WriteLine("Dzień dobry koleżanko!");  
    } } else {  
    Console.WriteLine("Witaj kolego!");  
}  
}
```

30 Instrukcja goto

Słowo kluczowe break nakazuje zakończenie instrukcji switch.
Istnieje także możliwość przeskoczenia do innej etykiety case lub default przy pomocy instrukcji goto. Oto przykład:



```
Miscellaneous Files
int Mandat = 50;
switch (Mandat)
{
    case 10:
        Console.WriteLine("10 zł mogę zapłacić");
        goto case 20;
    case 20:
        Console.WriteLine("Oj, 20 zł to troszkę dużo");
        break;
    default:
        Console.WriteLine("Tyle nie płacę!");
        goto case 10;
}
```

31 Pętle

Jednym z najczęściej używanych rodzajów pętli w języku C# jest pętla while. W pętlach while, tak jak w instrukcjach if, posługujemy się operatorami logicznymi oraz operatorami porównania.

Ogólna budowa pętli while w C# jest następująca:

```
while ( warunek zakończenia )  
{  
    // kod do wielokrotnego wykonania  
    //w tym inkrementacja zmiennej warunkowej  
}
```


32 Pętla do-while

Pętla do-while jest bardzo podobna do pętli while z tym, że jest ona zawsze wykonywana co najmniej raz, ponieważ warunek jej zakończenia jest sprawdzany po wykonaniu jednego jej przebiegu. Przykład pętli do while:

```
do {  
    Console.WriteLine("Pętla do-while, to się wykona");  
} while (X < 20);
```

Można powiedzieć że pętla do while jest podobna do pętli repeat w języku Pascal.

33 Pętla for

Innym popularnym rodzajem pętli jest pętla for. Jej użycie jest podobne jak w języku Java.

Przykład prostej pętli for:

```
for (int i = 1; i <= 10; i++) {  
    Console.WriteLine("Odliczanie..." + i);  
}
```

34 Parametry opcjonalne pętli for

Poszczególne informacje w nagłówku pętli są oddzielone od siebie znakiem średnika. Pętla for w języku C# jest na tyle elastyczna, że pozwala na pominięcie dowolnych elementów nagłówka. Na przykład:

```
for (int i = 20; i > 0; ) {  
    Console.WriteLine("Odliczanie..." + i);  
    i -= 2;  
}
```

Licznik pętli zmniejszyliśmy w jej ciele. W nagłówku ta informacja została pominięta.

Istnieje możliwość pominięcia jakiegokolwiek elementu z nagłówka np.:

```
for ( ; ; )
```

W ten sposób tworzymy tzw. pętlę forever, która będzie wykonywana w nieskończoność.

W C# istnieje też inna pętla tzn. foreach. Można jej używać w kolekcjach.

35 Instrukcja break

Słowo kluczowe break może być użyte do tzw. wyskoku z pętli.

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int i = 0;
        for (;;)
        {
            ++i;
            Console.WriteLine("Odliczanie..." + i);
            if (i == 20) { break; }
        }
        Console.Read();
    }
}
```

36 Instrukcja continue

Instrukcja break umożliwia natychmiastowe wyjście z pętli, natomiast instrukcja continue pozwala na przeskoczenie do następnej iteracji.

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int i = 0;
        for (;;)
        {
            ++i;
            // pomijamy wartości parzyste
            if (i % 2 == 0)
            {
                continue;
            }
            Console.WriteLine("Odliczanie..." + i);
            // pomijamy wartości parzyste,
            // warunkiem jest osiągnięcie wartości 101
            if (i == 101)
            {
                break;
            }
        }
        Console.Read();
    }
}
```

37 Operator warunkowy

Operator warunkowy ?: przypomina instrukcję if-else. Stosuje się go w prostych warunkach w celu zmniejszenia ilości kodu. Jego budowa jest następująca:

```
warunek ? (pierwsze wyrażenie) : (drugie wyrażenie);
```

Jeżeli warunek zostanie spełniony, rezultatem takiej operacji będzie wynik wyrażenia umieszczonego po znaku zapytania. W przeciwnym wypadku rezultatem operacji będzie wynik wyrażenia umieszczonego po znaku dwukropka.

38 Operator warunkowy

Przykład zastosowania operatora warunkowego:

```
string S;  
S = 10 > 20 ? "Nie wyświetlone" : "Wyświetlone";  
Console.WriteLine(S);
```

W warunku tej konstrukcji można stosować operatory logiczne, tak jak w zwykłej instrukcji if na przykład:

```
string S, S1, S2;  
int X, Y;  
X = 10;  
Y = 10;  
S1 = "Tak";  
S2 = "Nie";  
S = (X > 5 && Y == 10) ? S1 : S2;  
Console.WriteLine(S);  
//zostanie wyświetlony łańcuch s1
```

39 Konwersja typów

Konwersja typów to przekształcanie danych jednego typu na inny np. string na integer. W przypadkach gdy typy są niekompatybilne musimy użyć do ich konwersji specjalnych metod. w .NET Framework znajdują się one w klasie Convert.

Przykład użycia klasy Convert:

```
string S;  
int I;  
S = "18";  
I = Convert.ToInt32(S);  
I *= 2;  
S = Convert.ToString(I);  
Console.WriteLine(S);
```


40 Metody konwertujące

W klasie `convert` mamy do dyspozycji następujące metody konwertujące:

- ★ `ToBoolean()` - Konwertuje na typ `bool`.
- ★ `ToByte()` - Konwertuje na typ `byte`.
- ★ `ToChar()` - Konwertuje na pojedynczy znak.
- ★ `ToInt16()` - Konwertuje wartość na 16-bitową liczbę całkowitą.
- ★ `ToInt32()` - Konwertuje wartość na 32-bitową liczbę całkowitą.
- ★ `ToInt64()` - Konwertuje wartość na 64-bitową liczbę całkowitą.
- ★ `ToSingle()` - Konwertuje wartość na liczbę zmiennoprzecinkową.
- ★ `ToString()` - Konwertuje dane na łańcuch typu `string`.

41 Rzutowanie typów

Rzutowanie daje możliwość konwersji różnych typów danych.

Przykład nieprawidłowego przypisania

```
char C;  
byte B;  
C = 'A';  
B = C;
```

Taki kod nie zostanie prawidłowo skompilowany, gdyż próbujemy do zmiennej typu `byte` przypisać wartość zmiennej `char`. Aby wykonać takie przypisanie, możemy wykorzystać mechanizm rzutowania w następujący sposób:

Przykład rzutowania

```
B = (byte)C;
```

Po wykonaniu takiego kodu zmienna `B` będzie zawierała wartość 65, co odpowiada numerowi ASCII litery `A`.

42 Operacje na łańcuchach

Łańcuchy znakowe są jednym z częściej używanych typów danych. Najczęściej używaną operacją na łańcuchach jest konkatencja. W C# podobnie jak w Java możemy do niej użyć operatora +.

Przykład konkatencji:

```
class KlasaGlowna {  
    static void Main() {  
        string napis;  
        napis = "Pierwszy " + "Drugi";  
        System.Console.WriteLine(napis);  
        System.Console.ReadLine();  
    } }
```

43 Operacje na łańcuchach

Za pomocą operatora „+” możemy także łączyć zmienne typu string z innymi zmiennymi lub jawnie podanymi łańcuchami. Na przykład:

```
class KlasaGlowna {  
    static void Main() {  
        string napis1 = "Pierwszy napis";  
        string napis2 = "Drugi napis";  
        string zlaczonyNapis = napis1 + napis2;  
        System.Console.WriteLine(zlaczonyNapis);  
        System.Console.ReadLine();  
    } }
```

44 Łączenie łańcuchów z użyciem parametrów

Używając metody "WriteLine()" możemy łączyć łańcuchy w bardziej „wyrafinowany” sposób. Jako jej parametry możemy podać łańcuch oraz inne dane, które chcemy połączyć z poprzedzającym je łańcuchem.

```
class KlasaGlowna
{
    static void Main()
    {
        string napis1 = "pierwszy";
        string napis2 = "drugi";
        int ilosc = 3;
        System.Console.WriteLine("Ten łańcuch zawiera { 0} oraz { 1} parametr. ~
        Wszystkich parametrów jest { 2}.", napis1, napis2, ilosc);~
        System.Console.ReadLine();
    }
}
```

45 Formatowanie łańcuchów znakowych

Oprócz wstawiania parametrów w odpowiednie miejsce łańcucha, możemy je także formatować dodając do numeru parametru odpowiedni znak formatujący. Można używać następujących znaków formatujących:

- ★ C - waluta,
- ★ D - liczba dziesiętna, określa minimalną ilość cyfr, a brakujące wypełnia zerami,
- ★ E - liczba w notacji wykładniczej,
- ★ F - formatowanie z ustaloną liczbą miejsc po przecinku,
- ★ G - ogólne formatowanie,
- ★ N - podstawowy format liczbowy,
- ★ X - format heksadecymalny.

46 Przykład formatowania łańcuchów

```
class KlasaGlowna {  
    static void Main() {  
        System.Console.WriteLine("Formatowanie z C: {0:C}", 777.777);  
        System.Console.WriteLine("Formatowanie z D2: {0:D2}", 777);  
        System.Console.WriteLine("Formatowanie z D9: {0:D9}", 777);  
        System.Console.WriteLine("Formatowanie z E: {0:E}", 777.7777);  
        System.Console.WriteLine("Formatowanie z F2: {0:F2}", 777.7777);  
        System.Console.WriteLine("Formatowanie z F9: {0:F9}", 777.777);  
        System.Console.WriteLine("Formatowanie z G: {0:G}", 777.777);  
        System.Console.WriteLine("Formatowanie z N: {0:N}", 777.777);  
        System.Console.WriteLine("Formatowanie z X: {0:X}", 7779);  
        System.Console.ReadLine();  
    }  
}
```

47 Znaki specjalne w łańcuchach

Inne ważne znaki specjalne, których można używać tworząc napisy:

- ★ \a - „dzwonek”,
- ★ \b - „backspace”,
- ★ \r - powrót karetki,
- ★ \n - przejście do nowej linii,
- ★ \t - tabulacja w poziomie,
- ★ \v. - tabulacja w pionie,
- ★ \' - znak cudzysłowu
- ★ \\ - znak „backslash”,

48 Tablice

Tablice w C# (podobnie jak w innych językach) to zbiory wielu zmiennych tego samego typu. Innym podobnym złożonym typem są kolekcje.

W tablicach i kolekcjach możemy przetrzymywać nie tylko dane typów prostych takie jak liczby czy łańcuchy, ale także klasy i struktury. Dokładniej mówiąc, tablica lub kolekcja jest obiektem zawierającym zbiór referencji do innych zmiennych. W celu uproszczenia ich użycia wprowadzono mechanizm polegający na tym, że programista nie musi tego jawnie wskazywać tak jak to miało miejsce np w języku Pascal.

49 Użycie tablic

Aby użyć zmiennej tablicowej w C# powinniśmy zadeklarować typ przechowywanych w niej elementów oraz maksymalną dopuszczalną ich ilość.

Nowy obiekt tablicy tworzymy przy pomocy operatora "new".

Do konkretnego elementu tablicy odwołujemy się tak jak w Java, podając nazwę tablicy oraz indeks elementu w nawiasach kwadratowych.

Elementy tablic w C# są indeksowane od 0.

50 Przykład użycia tablic

```
using System;
using System.Collections.Generic;
using System.Text;
class KlasaGlowna
{
    static void Main()
    {
        int[] tablicaLiczby = new int[3];
        tablicaLiczby[0] = 11;
        tablicaLiczby[1] = 22;
        tablicaLiczby[2] = 33;
        System.Console.WriteLine("Liczby znajdujące się w tablicy:");
        System.Console.WriteLine(tablicaLiczby[0] + " " + tablicaLiczby[1] + " " + tablicaLiczby[2]);
        System.Console.WriteLine("Te same liczby ale wypisane w innej kolejności: ");
        System.Console.WriteLine(tablicaLiczby[1] + " " + tablicaLiczby[2] + " " + tablicaLiczby[0]);
        System.Console.Read();
    }
}
```

51 Wartości domyślne w tablicach

W języku C# w czasie tworzenia nowego obiektu tablicy, wszystkie jej elementy są wypełniane wartościami domyślnymi. np. dla liczb typu „int” jest to wartość 0, dla zmiennych typu „string” - łańcuch pusty, dla innych obiektów - „null” itd.

52 Struktury

Innym sposobem reprezentacji danych używanym w języku C# są struktury. Struktury w swojej koncepcji są bardzo podobne do klas. Podobnie jak klasy, zawierają one dane składowe oraz składowe funkcyjne. Struktura może zawierać także konstruktory, właściwości, metody, pola, a nawet typy zagnieżdżone. Najważniejszą rzeczą różniącą struktury od klas jest to, że struktur nie możemy dziedziczyć. W strukturach nie można także używać destruktorów. Ponadto, struktura jest typem skalarnym, natomiast klasa typem referencyjnym. Z tego powodu struktur powinno się używać do definiowania niewielkich i prostych typów. Są one przydatne do reprezentowania obiektów, które nie wymagają współpracy z typami referencyjnymi.

53 Struktury

Struktury definiujemy podobnie jak klasy zastępując słowo class słowem struct.

```
struct Punkt
{
    //definiujemy składowe
    public int x;
    public int y;
    //definiujemy konstruktor
    public Punkt(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}
```

54 Wykorzystanie struktur

Użycie zdefiniowanej wyżej struktury punkt:

```
class Pokaz
{
    public static void Main()
    {
        Punkt a = new Punkt(10, 2);
        Punkt b = new Punkt(-4, 3);
        Console.WriteLine("Punkt 'a' ma współrzędne: x = {0}, y = { 1} ", a.x, a.y);
        Console.WriteLine("Punkt 'b' ma współrzędne: x = {0}, y = { 1} ", b.x, b.y);
        a = b;
        b.x = 33;
        b.y = -125;
        Console.WriteLine("\nZamiana");
        Console.WriteLine("Punkt 'a' ma współrzędne: x = {0}, y = { 1} ", a.x, a.y);
        Console.WriteLine("Punkt 'b' ma współrzędne: x = {0}, y = { 1} ", b.x, b.y);
    }
}
```

55 Struktury z akcesorami

W strukturach możemy używać tzw. akcesorów. Akcesory są to specjalne metody, które udostępniają nam pola struktury zarówno do odczytu jak i do zapisu. Mają one nazwy get i set. Z powodu ich funkcji przy ich implementacji używamy nieco innej składni jak w przypadku zwykłych metod.

W strukturach możemy także implementować zwykłe metody.

56 Struktury z akcesorami

```
public struct Samochody {  
    private string samochod1;  
    private string samochod2;  
    public Samochody(string samochod1, string samochod2) {  
        this.samochod1 = samochod1;  
        this.samochod2 = samochod2;  
    }  
    public string Samochod1 {  
        get { return samochod1; }  
        set { samochod1 = value; }  
    }  
    public string Samochod2 {  
        get { return samochod2; }  
        set { samochod2 = value; }  
    }  
    public override string ToString() {  
        return (String.Format("{0} oraz {1}", samochod1, samochod2));  
    }  
}
```

57 Użycie struktury z akcesorami

```
class Glowna {  
    static void Main() {  
        Samochody s;  
        s.Samochod1 = "Fiat";  
        S.Samochod2 = "Kia";  
        System.Console.WriteLine("Nie lubie takich samochodów  
        jak: {0}.", s);  
    }  
}
```

58 Egzemplarze struktury

Aby stworzyć egzemplarz struktury należy użyć słowa kluczowego `new`. Taki nowy egzemplarz w konstruktorze otrzymuje wymagane przez ten konstruktor wartości. Ponieważ struktura jest typem skalarnym, wywołując metodę operującą na strukturze przekazujemy do niej konkretne wartości a nie referencję. Oznacza to, że za każdym nowym wywołaniem metody tworzona jest kopia struktury i na niej wykonywane są operacje.

Gdybyśmy do implementacji powyższego programu użyli klas, za każdym razem wywołanie metody `mojaMetoda` wykonywałoby operacje na oryginalnych danych klasy a nie na kopii.

59 Powoływanie nowej struktury

Nowe struktury można tworzyć także bez użycia słowa kluczowego `new`. Można to zrobić w następujący sposób:

```
class Główna {  
    static void Main() {  
        Samochody s;  
        s.Samochod1 = "Fiat";  
        s.Samochod2 = "Kia";  
        System.Console.WriteLine("Nie lubię takich samochodów  
        jak: {0}.", s);  
    }  
}
```