

# C# DLA BLONDYNKI, CZYLI O PROGRAMOWANIU INACZEJ.

Angelika Maria Piątkowska



# Spis treści

<b>1. Wstęp</b>	<b>1</b>
1.1. Co to za książka i dlaczego powstała? . . . . .	1
1.2. Kto ją pisze? . . . . .	3
1.2.1. Angelika Maria Piątkowska . . . . .	3
1.3. O czym będzie? . . . . .	3
<b>2. Hello world</b>	<b>4</b>
2.1. Kod na start . . . . .	4
2.2. Funkcja . . . . .	6



# Spis rysunków



Spis tabel





# Rozdział 1

## Wstęp

### 1.1. Co to za książka i dlaczego powstała?

No i powstaje książka. Mam nadzieję, że nie będzie to kolejna nudna książka rozpoczynająca się od kilku rozdziałów o historii, kilku kolejnych o tym jakie są typy itd, by na 70 stronie pokazać jak zbudować projekt a dalej zapoznawać czytelnika systematycznie z kolejnymi częściami języka i platformy. Jest to oderwane od prawdziwego programowania na tyle, że ciężko to zastosować w momencie kiedy już ten projekt się zbudować potrafi. Kto nie przeczyta i tak sobie znajdzie dowolny kurs video i obejrzy pierwszy odcinek, aby umieć to samo:) Prawda jest taka, że początkujący, który nie wie co to "if" nie musi wiedzieć co to BCL, CIL (albo IL) czy GAC. To z czasem przyjdzie i czytanie o tym na początku ma taki sam sens, jak wkuwanie bandplanów<sup>1</sup> na egzamin radioamatorski czy warstwy tcp/ip na egzamin z sieci - słowem: po egzaminie i tak się zapomina po tygodniu i długo jeszcze się nie pamięta, by potem stało się to oczywistością. Ale aby się stało potrzeba trochę doświadczenia, którego w momencie egzaminu czy czytania książki dla początkującego oczywiście nie ma.

Co więc potrzebujemy na początek?

Na początku było słowo. W naszym przypadku byłoby to słowo kluczowe, a szerzej kod. Więc zawsze to od kodu będziemy zaczynać, aby następnie poopowiadać sobie na jego temat wchodząc w dygresję tak daleko, na ile wyobraźnia pozwoli, two-

---

<sup>1</sup>Dozwolone częstotliwości na których można nadawać

rzając "haczyki do wiedzy". Zaczniemy od szczątkowego, bądź co bądź, doświadczenia, aby już od początku każdy nowy kawałek wiedzy kojarzył nam się z czymś, co możemy zmaterializować w postaci ciągu znaków potrafiącego się skompilować do ILA<sup>2</sup> i wykonać. Takie podejście wymusi inną niż standardowa kolejność przyswajania materiału, co pozwoli osobie uczącej się rozpocząć swoją przygodę z programowaniem.

Całość jednak będzie nastawiona na ciekawskich - chcących wiedzieć jak najwięcej o tym jak działa ich kod. Jak to pod spodem się przelicza. Zejść tak nisko jak się da i mieć świadomość tego co się pisze. Myślę, że wiele osób zaawansowanych również będzie miało okazję dowiedzieć się czegoś nowego.

Nie przeraż się drogi czytelniku ilością odnośników. Linki w dokumencie są klikalne nie bez powodu. Uznałam bowiem, że jeżeli coś jest gdzieś dobrze opisane, to wystarczy dowiązać do tego bogactwa. Przepisywanie nie ma najmniejszego sensu.

Książka w wersji elektronicznej jest dostępna za darmo, wraz z kodem źródłowym (tak, książka też posiada swój kod źródłowy - takie pliki .tex). Jest w formacie A4, który ma ułatwić osobom drukowanie wybranego fragmentu na domowej drukarce. Można też wydrukować całość i oprawić, jak ktoś chce. Taka forma wydaje mi się najbardziej w porządku, ponieważ kiedy zaczynałam naukę, to również starałam się uczyć z darmowych materiałów, lub wypożyczonych czy skopiowanych w ramach dozwolonego użytku. Nie było mnie wówczas stać na poszerzanie biblioteczki. Kiedy już zaczęłam pracę, zaczęła się ona poszerzać szybciej niż moje moce przerobowe i zaczęłam spłacać dług wdzięczności dla autorów książek. Teraz tą pozycją chcę spłacić dług względem społeczności. Moim marzeniem jest, by ten, kto chce się uczyć nie musiał lawirować na granicy prawa pożyczając książki od nieznajomych, nierzadko obywatelstwa chińskiego czy rosyjskiego, dlatego w imię wolności informacji ową wiedzę tu przekazuję.

Zezwalam na kopiowanie, przerabianie, wrzucanie gdzie popadnie- github ma informacje kto to napisał i kiedy, więc jeżeli ktoś będzie potrzebował dotrzeć do źródła, to dotrze. Repozytorium dostępne jest pod adresem [https://github.com/Piatkosia/cs\\_blondi](https://github.com/Piatkosia/cs_blondi).

---

<sup>2</sup>Taki język pośredni, coś, co jest w zależności od potrzeb kompilowane do assemblera

## 1.2. Kto ją pisze?

Na razie podpisałam siebie. Jeżeli pojawi się jakiś "kontrybutor", to podmienię na "praca zbiorowa" a kolejna osoba napisze coś o sobie poniżej, w podsekcji.

### 1.2.1. Angelika Maria Piątkowska

Założycielka #listekklonu, 2 lata doświadczenia zawodowego jako programistka, organizatorka Security BSides Warsaw. Interesują mnie różne aspekty bezpieczeństwa, od safety po security. Członkini LOK, sympatyczka POC. Lubię czytać dobrą literaturę i czasami sama coś napiszę do szuflady.

## 1.3. O czym będzie?

Zacniemy od hello world, bo inaczej się nie da, aby następnie przejść przez ....  
(in progress)

# Rozdział 2

## Hello world

O samym "hello world" można napisać bardzo dużo. Wbrew pozorom dużo się dzieje, aby ten program został wypisany.

### 2.1. Kod na start

Poniżej macie sławnego hello worlda. Nazywa się tak program, jaki zwykle koder pisze, kiedy zaczyna pracę z nowym językiem. Całość polega na wypisaniu tekstu "hello world" na standardowe wyjście, które domyślnie jest definiowane jako konsola.

---

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace PierwszyProgram
8 {
9     class Program
10    {
11        static void Main(string[] args)
12        {
13            Console.WriteLine("Hello world");
14            Console.ReadKey();
15        }
16    }
```

---

17 }

---

Tak na prawdę napisałam tylko linię 13 i 14, reszta została dla mnie wygenerowana przez projekt konsolowy. Pierwsze co się rzuca w oczy, to using. Kilka z nich jest na szaro, a to oznacza, że możemy je bez problemów usunąć. Mamy teraz coś takiego:

---

```
1 using System;
2 namespace PierwszyProgram
3 {
4     class Program
5     {
6         static void Main(string[] args)
7         {
8             Console.WriteLine("Hello world");
9             Console.ReadKey();
10        }
11    }
12 }
```

---

Został nam tylko jeden using. Ale na upartego i jego możemy usunąć, acz musimy nieco zmodyfikować kod, aby się on skompilował.

---

```
1 namespace PierwszyProgram
2 {
3     class Program
4     {
5         static void Main(string[] args)
6         {
7             System.Console.WriteLine("Hello world");
8             System.Console.ReadKey();
9         }
10    }
11 }
```

---

Można całość napisać jeszcze inaczej. Tak trochę dla fanów nowinek technicznych, bo kiedyś nie można tak było napisać.

---

```
1 using static System.Console;
2 namespace PierwszyProgram
3 {
4     class Program
```

```
5    {  
6        static void Main(string[] args)  
7    {  
8        WriteLine("Hello world");  
9        ReadKey();  
10    }  
11 }  
12 }
```

---

Z ciekawych elementów jakie można tu opisać należy wymienić:

- funkcja,
- wywołanie funkcji,
- słowo kluczowe using,
- słowo kluczowe namespace,
- klasa,
- słowo kluczowe static,
- standardowe wejście i wyjście - konsola,
- funkcja Main,
- entry point programu,
- kompilacja i uruchomienie.

Także tym (niekoniecznie w tej kolejności) się będziemy spokojnie mogli zająć w pierwszym rozdziale.

## 2.2. Funkcja

Program bez funkcji by nie istniał, bo po co tworzyć program, który nic nie robi. Każdy z nich ma jakąś funkcję, chyba że jedyną funkcją programu jest to, że ma się uruchomić;) Ale aby się uruchomił, musi być jakaś funkcja, no nie da się inaczej. W przypadku C# ta funkcja musi być umieszczona w jakiejś klasie,

ale nie wyprzedzajmy faktów. Zaczniemy od tego, w jaki sposób funkcję można wywołać. Najprościej można przedstawić jako `Funkcja(parametry)`; gdzie nazwa jest nazwą funkcji, zaś parametry są to informacje jakie podajemy funkcji. Na końcu jest średnik. Średnik mówi programowi, że skończyliśmy jeden "krok" programu. Kiedy chcemy poznać wynik działania naszej funkcji, to (jeżeli zwraca wartość) robimy

```
wynik = Funkcja(parametry);
```

Kiedy mamy wiele parametrów, to je rozdzielamy przecinkami. Jak znamy nazwę możemy ją podać przed dwukropkiem. Co nam da

```
wynik = Funkcja(parametr1, parametr2, parametr3);
```

albo

```
wynik = Funkcja(par1:parametr1, par2:parametr2, par3:parametr3);
```

Tak, wygląda trochę jak funkcja matematyczna i nic dziwnego. Ów zapis w językach c-like był właśnie na tym wzorowany.

To co może zainteresować bezpieczników czy miłośników optymalizacji, to kolejność przekazywania parametrów na stos. Programiści c++ mają dość spory wybór, o czym można przeczytać na msdn pod adresem <https://msdn.microsoft.com/pl-pl/library/984x0h58.aspx> my, programiści c# niekoniecznie. Jak można przeczytać w [1] "During the run-time processing of a function member invocation (§7.5.4), the expressions or variable references of an argument list are evaluated in order, from left to right, (...)" czyli zawsze mamy od lewej do prawej, czyli taki c++owy `__cdecl`. Więcej można przeczytać na <https://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf>. Najlepiej zapisz sobie ten plik, bo będziemy do niego często wracać.

To teraz spójrzmy na deklarację metody. Najogólniej byłoby to:

[atrybut] specyfikator\_dostępu rodzaj\_wartości\_zwracana nazwa(parametry) Tu wpisz jakiś kod;

Zaznaczę tylko, że wielkość liter (o ile nie napisano inaczej) ma znaczenie, więc funkcja o nazwie `GetItem()` i `getItem()` to dwie zupełnie inne funkcje.

Można elementy pomijać, jednak kolejność ich zawsze musi być taka jak powyżej. Atrybutami nie będziemy się teraz zajmować, dociekliwych odsyłam do [http:](http://)

`//plukasiewicz.net/Csharp_dla_zaawansowanych/Atrybuty` gdzie można o tym poczytać. Specyfikatory dostępu to kolejno:

**public** Można wywołać metodę wszędzie (oczywiście na zewnątrz assembly trzeba dodać referencję w projekcie)

**private** Można wywołać wyłącznie w tej samej klasie, w której jest metoda. Przydatne przy enkapsulacji<sup>1</sup>.

**protected** Można wywołać w klasie, oraz jej dzieciach (klasach dziedziczących po...).

**internal** Można wywołać z poziomu tego konkretnego assembly

**protected internal** Zlepka dwóch poprzednich. Czyli jak w tym samym assembly to zewsząd, jak w innym, to tylko klasy dziedziczące.

Rodzaj, w tym schemacie to słówka takie jak `static` (wywoływane na rzecz klasy, nie obiektu), `override` (nadpisanie funkcji), `new` (przysłonięcie funkcji), `virtual` (deklaracja funkcji którą można nadpisać), `abstract` (funkcja bez ciała w klasach abstrakcyjnych) itd.

wartość zwracana, to wynik funkcji. Jeżeli nie zwracamy nic, wpisujemy `void`. Jeżeli zwracamy, to wpisujemy tutaj odpowiedni typ- albo predefiniowany, albo swój, ale typ musi istnieć. Przykłady typów predefiniowanych to `bool` (prawda/fałsz), `int` (liczba całkowita), `float` (liczba zmiennoprzecinkowa), `string` (napis). Pełna ich lista znajduje się na <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/built-in-types-table>

O parametrach napisałam wyżej. Dodam, że domyślnie są one przekazywane przez wartość w przypadku typów wartościowych (value type) oraz przez referencję w przypadku typów referencyjnych (reference type). Jeżeli chcemy zaś zawsze przekazać przez referencję, to przed parametrem dajemy `ref` lub `out`. `Out` nam inicjalizuje po wejściu, a `ref` używa aktualnej wartości (więc musimy zainicjalizować taką zmienną przed jej wysłaniem do funkcji).

Wracając do funkcji abstrakcyjnej (zwanej inaczej czysto wirtualną) to jest ona trochę śmieszna. Jej przykład to:

---

<sup>1</sup>Przyda się jak będziemy omawiać programowanie obiektowe



```
public abstract void ZimplementujMnie();
```

czyli średnik od razu po nawiasach z parametrami, żadnych klamerek.

Ok, to już deklarację każdy będzie umiał przeczytać. Na razie o funkcjach wystarczy, z pewnością wrócimy jeszcze do tego tematu.

# Bibliografia

- [1] ECMA-334. *C# Language Specification*. ECMA 334, June 2006.