**[KURS C#] Wartość zmienną jest**



**UWAGA!**

**Nowa wersja kursu dostępna** jest na moim portalu z kursami. Nadal jest on **BEZPŁATNY**! A jednocześnie posiada więcej lekcji, wygodną nawigację i możliwość zapamiętania, w którym miejscu skończyłeś ostatnio!. Znajdziesz go [**POD TYM LINKIEM**](https://kurs-szarpania.pl/nauka/product/szkola-szarpania-intro)!

Projekt aplikacji dla Jacka nabiera rozpędu. Dodaliśmy już nasze **pierwsze klasy**, które reprezentują dwa rodzaje kont. Jednak są one jakieś takie puste. A przecież konta powinny zawierać jakieś **informacje**, bo jak inaczej poznamy do kogo należą? Otóż to! Także teraz porozmawiamy o **przechowywaniu danych**.

Wszystkie programy komputerowe operują na jakichś informacjach, na danych. Jednak żeby to było możliwe muszą mieć jakiś sposób ich przechowywania i przekazywania w różne miejsca systemu. W tym celu istnieją w kodzie m.in. **zmienne**. To pojęcie poznaliśmy przy okazji omawiania teorii. Zmienne są jak **pudełka** przeznaczone do przechowywania konkretnego modelu jakiegoś urządzenia kiedy pakujemy je do wysyłki.

**Plan i wykonanie**

Na początek przyjmijmy, że istnieją **dwa rodzaje zmiennych** – te, które umieszczamy w klasie i te, których używamy w funkcjach.

Nie jest to żaden oficjalny podział. Przyjmujemy go tutaj dla wyjaśnienia pewnych zagadnień.

Pierwsze z nich są jakby **fragmentem schematu** zaznaczonym ramką i podpisanym „tutaj będzie zasilanie urządzenia”.

Drugie zaś są istniejącymi pudełkami, do których coś możemy zapakować. Te zmienne, które znajdują się w klasie będą potem wydzielonym fragmentem wewnątrz obudowy urządzenia, do którego wstawiamy poszczególne części.

**Własne miejsce**

Wyobraź sobie, że projektujesz obudowę do komputera. Plan obudowy jest klasą. W obudowie wydzielasz poszczególne miejsca na części, takie jak miejsce na zasilacz czy płytę główną. **Te miejsca opisane na schemacie to będą nasze zmienne w klasie**, nazywane są **polami klasy**.

Kiedy wyprodukujemy już faktyczną obudowę na podstawie tego schematu (**klasy**) to będziemy w niej mieli wyznaczone wcześniej miejsca (**zmienne**), do których teraz możemy wstawić określone części.

Oprócz tego mamy pudełko na tą naszą obudowę, które niesiemy ze sklepu. To pudełko to jest **zmienna w funkcji** (funkcji niesienia).

**Potrzymaj mi coś**

Skoro już wiemy o co chodzie ze zmiennymi to dodajmy w końcu jakieś w kodzie.

Otwórz projekt banku, utworzony wcześniej.

Przejdźmy do pliku ***Program.cs***. Dopisaliśmy w nim wyświetlanie nazwy aplikacji i autora. Jednak teraz jest to zrobione tak jakbyśmy wzięli jakieś urządzenie w fabryce i wrzucili do sklepu bez żadnego pakowania. Zmieńmy to i **zapakujmy oba teksty w zmienne**.

Kod przed zmianami, który poprzednio zostawiliśmy, wygląda tak:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Bank

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Nazwa: Bank");

Console.WriteLine("Autor: Marek Zajac");

Console.ReadKey();

}

}

}

Czas na modyfikację. Nad linijkami z WriteLine() dodajmy **dwie zmienne**:

namespace Bank

{

class Program

{

static void Main(string[] args)

{

string name;

string author;

Console.WriteLine("Nazwa: Bank");

Console.WriteLine("Autor: Marek Zajac");

Console.ReadKey();

}

}

}

W ten sposób dodajemy **zmienne w naszym kodzie**. Składają się one z **dwóch części** – **typu** i **nazwy** unikalnej w danym bloku (bloki to te fragmenty kodu pomiędzy otwierającym i zamykającym nawiasem klamrowym).

Ale jakie znowu typy? O co chodzi?

**Tego typu**

Porównaliśmy zmienne do pudełek, które służą do przechowywania konkretnych obiektów. Powiedzmy, że mówimy o wspominanych w poprzednich częściach głośnikach komputerowych.

Typ, który podajemy przy dodawaniu zmiennej jest jak powiedzenie „to jest pudełko na głośnik typu SUPERBASS”.

Ale do fabryki składającej tego głośniki przychodzą **pomniejsze części**, którego mają swoje pudełka. Np. możemy mieć pudełko na wzmacniacz, który będzie elementem głośników. Tak samo jest z **typami zmiennych**.

Gdybyśmy mieli **klasę** opisującą nasz głośnik:

class Glosnik

{

}

I chcieli zrobić **pudełko** na niego to wyglądało by to w ten sposób:

Glosnik pudelkoNaGlosnik;

W ten sposób taka zmienna będzie mogą przechowywać obiekty klasy Głośnik.

**Dają to bierzemy**

Jednak nie zawsze korzystamy z typów jakie sami utworzyliśmy. Czasami, tak jak w przypadku nazwy i autora, potrzebujemy przechować powszechnie używany **typ danych**. Taki jak tekst.

W tym celu język C# daje nam do dyspozycji kilka **wbudowanych typów**. Ich listę znajdziesz na [stronie Microsoftu](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/built-in-types-table).

W naszym przypadku skorzystaliśmy z typu **string**, który pozwala przechowywać tekst.

Inne najczęściej używane typy to **int** dla liczb całkowitych, **double** lub **decimal** dla liczb zmiennoprzecinkowych (takich z wartościami dziesiętnymi) czy **bool** dla wartości prawda/fałsz (***true/false***). Na podanej stronie możesz też sprawdzić jakie np. maksymalne albo minimalne wartości mogą trzymać niektóre typy, chociażby liczbowe. Bo mają one swoje ograniczenia.

No więc w naszym kodzie utworzyliśmy **dwie zmienne dla tekstu**. Ale są one puste! Zmieńmy to.

**Wezmę to**

Samo napisanie

string name;

To tylko **definicja zmiennej**. Tworzymy taką zmienną i ona sobie jest. To tak jak pudełko, które poskładaliśmy. Jest. Ale jest puste.

Aby wypełnić nasze pudełko (zmienną) musimy **przypisać do niej jakąś wartość**. Zmienne wypełniany korzystając ze znaku równości.

Włóżmy zatem dwa nasze teksty do przygotowanych dla nich zmiennych:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Bank

{

class Program

{

static void Main(string[] args)

{

string name;

name = "Nazwa: Bank";

string author;

author = "Autor: Marek Zajac";

Console.WriteLine("Nazwa: Bank");

Console.WriteLine("Autor: Marek Zajac");

Console.ReadKey();

}

}

}

Wzięliśmy stworzone zmienne korzystając z nadanym im nazw, do tego skorzystaliśmy ze **znaku równości**. Bierze on wartość po prawej stronie i wrzuca ją do pudełka po lewej jego stronie. Od teraz w zmiennej name  znajduje się tekst ***„Nazwa: Bank”***, a zmienna author  zawiera tekst ***„Autor: Marek Zajac”***.

Jednak nie musimy wkładać wartości do zmiennych w **osobnych linijkach** (**instrukcjach**). Możemy to zrobić w momencie ich tworzenia. Będzie to **inicjalizacja**. Można to porównać do obudowywania urządzenia opakowaniem albo owijania go folią – **opakowanie powstaje w momencie pakowania**:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Bank

{

class Program

{

static void Main(string[] args)

{

string name = "Nazwa: Bank";

string author = "Autor: Marek Zajac";

Console.WriteLine("Nazwa: Bank");

Console.WriteLine("Autor: Marek Zajac");

Console.ReadKey();

}

}

}

Ważne żeby zapamiętać, że **nie możemy do zmiennej przypisać innego rodzaju wartości niż podany przez nas typ**.

Czyli do zmiennej typu **string** nie wpiszemy liczby, a do zmiennej typu **Głośnik** nie wstawimy tekstu.

**Niezdecydowany**

Istnieje jednak słowo, którego możemy użyć kiedy np. ten docelowy typ ma za długą nazwę albo z innego powodu nie chcemy go sami podawać tylko chcemy żeby go za nas ustawiło środowisko. Tym słowem jest słowo var .

Kiedy wpiszemy je jako **typ zmiennej** zostanie ono od razu podmienione na właściwy typ pierwszej przypisanej do takiej zmiennej wartości. Ograniczenie jest takie, że musimy w tym wypadku skorzystać z **inicjalizacji** bo od początku musi być **znany docelowy typ**, a nie jest to możliwe kiedy nic do tak stworzonej zmiennej nie wstawiliśmy od razu.

Wcześniejszy kod, ale z z użyciem słowa var  wyglądał będzie następująco:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Bank

{

class Program

{

static void Main(string[] args)

{

var name = "Nazwa: Bank";

var author = "Autor: Marek Zajac";

Console.WriteLine("Nazwa: Bank");

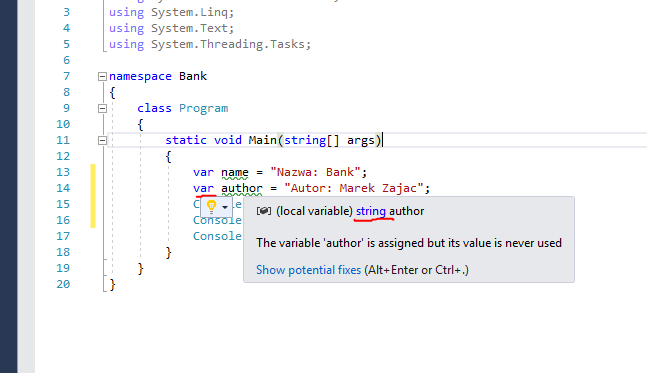
Console.WriteLine("Autor: Marek Zajac");

Console.ReadKey();

}

}

Kiedy najedziemy kursorem na **nazwę zmiennej** to zobaczymy jednak, że jest ona typu **string**, mimo, że nigdzie go jawnie nie podaliśmy:

[](https://zajacmarek.com/wp-content/uploads/2018/11/variables_3.png)

Używanie tego typu po pierwsze **ułatwia późniejsze zmiany w kodzie**, bo automatycznie dostosuje się do nowego typu jaki będziemy chcieli przypisać, a po drugie przydaje się kiedy nazwy typów są bardzo złożone (a w przyszłości poznamy „sposób” tworzenia takich złożonych typów).

Za to teraz przyszła pora na użycie stworzonych przez nas zmiennych.

**Weź mi to wydrukuj**

Mamy to szczęście, że funkcja WriteLine()  wie co zrobić z naszymi zmiennymi typu **string**. Dzięki temu możemy je przekazać tak jak wcześniejszy tekst:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Bank

{

class Program

{

static void Main(string[] args)

{

string name = "Nazwa: Bank";

string author = "Autor: Marek Zajac";

Console.WriteLine(name);

Console.WriteLine(author);

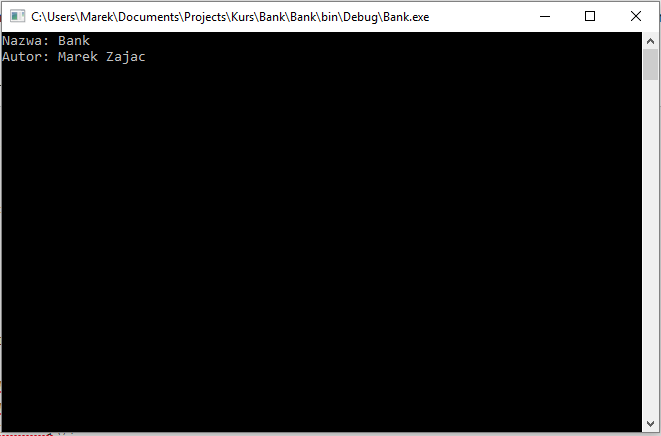
Console.ReadKey();

}

}

}

Uruchamiając program zobaczymy na ekranie to co poprzednio:

[](https://zajacmarek.com/wp-content/uploads/2018/11/first_program_text.png)

Wszystko nadal działa jak poprzednio, a my schowaliśmy nasz tekst do zmiennych, z których możemy go teraz **w razie potrzeby wyciągać**. Bo co jeżeli jakąś wartość musimy użyć kilka razy? Np. w jakimś działaniu? Wtedy zmienne pozwalają nam zapanować nad tym i w razie konieczności zmiany tej wartości uniknąć żmudnego szukania gdzie ona się pojawia.

Teraz kiedy będziemy potrzebowali w dwóch miejscach wydrukować nazwisko autora wystarczy, że jeszcze raz użyjemy **wartości trzymanej w zmiennej**:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Bank

{

class Program

{

static void Main(string[] args)

{

string name = "Nazwa: Bank";

string author = "Autor: Marek Zajac";

Console.WriteLine(name);

Console.WriteLine(author);

Console.WriteLine(author);

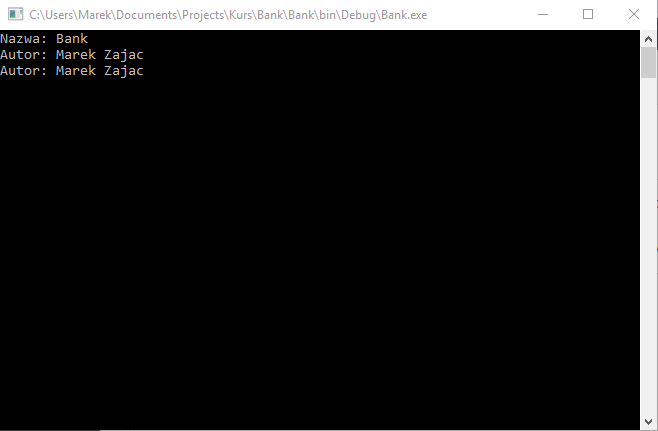
Console.ReadKey();

}

}

}

Uruchomienie tego programu da efekt pokazany na obrazku:

[](https://zajacmarek.com/wp-content/uploads/2018/11/variables_1.png)

Jeżeli mówimy o zmiennych to nazwa ta sugeruje, że powinniśmy móc je jakoś **modyfikować**, prawda?

**Zmienna zmienną jest**

Faktycznie zmienne mogą mieć **wymienioną zawartość**. Tak samo jak możemy z pudełka wyciągnąć jeden egzemplarz i zamiast niego wstawić inny. Zróbmy więc tak aby nasz autor zmienił się pomiędzy kolejnymi wyświetleniami:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Bank

{

class Program

{

static void Main(string[] args)

{

string name = "Nazwa: Bank";

string author = "Autor: Marek Zajac";

Console.WriteLine(name);

Console.WriteLine(author);

author = "Autor: Jacek Placek";

Console.WriteLine(author);

Console.ReadKey();

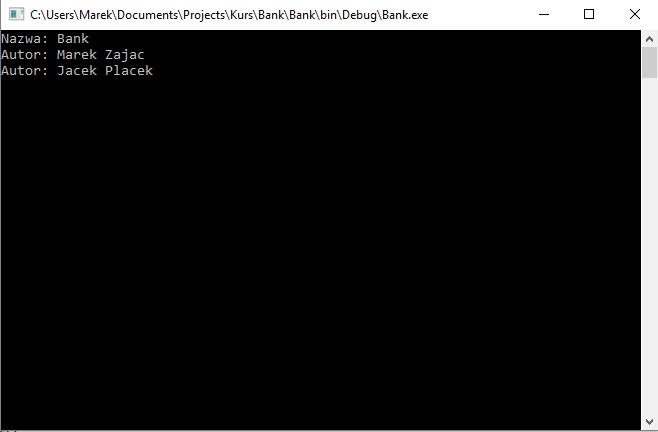
}

}

}

Skorzystaliśmy tutaj z takiego samego **znaku przypisania** jak przy nadawaniu pierwszej wartości.

Teraz uruchamiając program dostrzeżemy, że nasz kolega Jacek chyba już zaczyna uznawać nasz projekt za swoje dzieło…

[](https://zajacmarek.com/wp-content/uploads/2018/11/variables_2.png)

Wróćmy jednak do wersji, w której to my jesteśmy autorem i nasze imię wyświetla się tylko raz (tyle nam wystarczy).

Usuń więc drugie przypisanie autora do zmiennej i drugie jej wyświetlanie na ekranie.

namespace Bank

{

class Program

{

static void Main(string[] args)

{

string name = "Nazwa: Bank";

string author = "Autor: Marek Zajac";

Console.WriteLine(name);

Console.WriteLine(author);

Console.ReadKey();

}

}

}

**Z klasą**

Na początku wspomniałem, że zmienne mogą też znaleźć się w naszych klasach. Będą to wtedy **wydzielone przestrzenie na informacje** jakie mają się potem w obiektach takich klas znajdować. A tak się składa, że nasza aplikacja bankowa takie miejsca potrzebuje.

Pamiętasz klasy, które utworzyliśmy? Są one takie puste. A przecież z kontem związane są pewne informacje, które są uzupełniane podczas jego zakładania. Dlatego teraz dowiemy się jak takie przestrzenie w klasach dodać.

**Dane konta**

Weźmy na początek **konto rozliczeniowe**. W tym momencie jego klasa wygląda w ten sposób:

namespace Bank

{

class BillingAccount

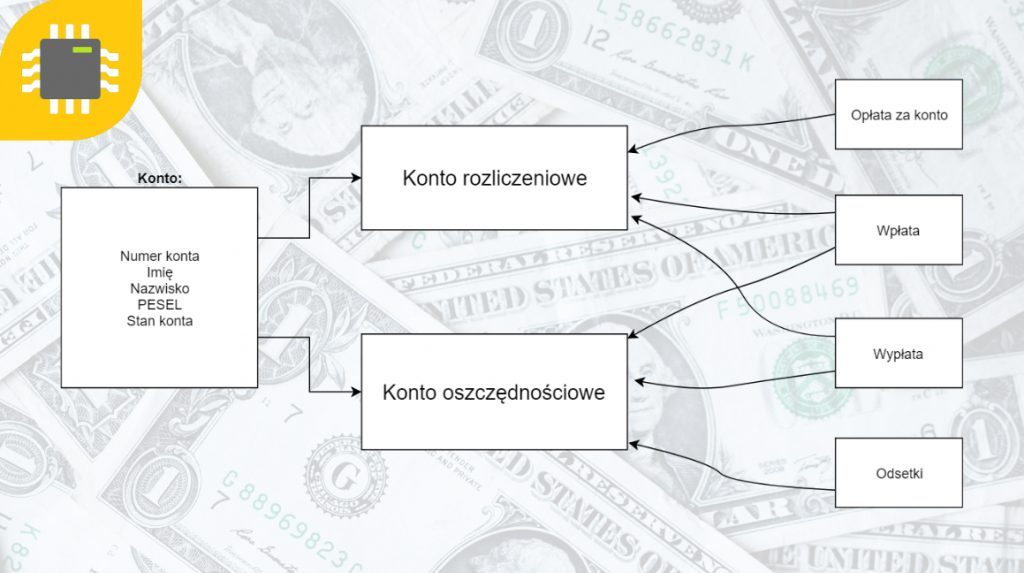
{

}

}

Dosyć biednie. A przecież bank nie może wyglądać biednie! Zróbmy coś z tym.

Przypomnij sobie diagram naszej aplikacji. Mamy tam widoczne **konto oszczędnościowe** i **rozliczeniowe**. Związane są one tym, że oba są kontami i zawierają pewne informacje:

[](https://zajacmarek.com/wp-content/uploads/2018/12/bank-diagram.jpg)

Tak więc takie miejsca na wartości potrzebujemy w naszych klasach. Skorzystamy więc z kilku dostępnych dla nas **typów danych**.

**Typowo**

Będziemy używać w tym wypadku typów:

* string dla **imienia**, **nazwiska** i **numer konta**. Dlaczego numer konta dajemy **jako tekst**? Bo jest długi, a do tego nie będziemy na nim wykonywać żadnych operacji typu dodawanie czy mnożenie. Dodatkowo jeżeli przypomnimy sobie wymagania z części gdzie planowaliśmy co projekt ma robić to znajdziemy tam pewną informację o numerze konta właśnie. Ma być on uzupełniany zerami w pewnych miejscach. taka operacja będzie przyjemniejsza kiedy będzie to po prostu tekst, do którego mamy wstawić jakiś znak.
* long dla numeru **PESEL**. Nie możemy użyć typu int bo 11 cyfrowa liczba nie zmieściłaby się w nim.
* decimal  dla stanu konta. Jeżeli przejrzałeś listę dostępnych standardowo typów to być może zauważyłeś, że dla liczb zmiennoprzecinkowych są jeszcze dostępne float  i double . Jednak decimal  ma **największą precyzję**, dzięki temu wykonywanie wielu operacji, nawet na dużych liczbach, nie spowoduje, że gdzieś zgubimy grosze naszych klientów.

Typy liczbowe posiadają pewne ograniczenia. Mają one swój limit w postaci minimalnej i maksymalnej wartości jaką potrafią zapamiętać. Informacje o tym jakie są to wartości możemy znaleźć chociażby w podanej wcześniej [dokumentacji Microsoftu](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/built-in-types-table). Kiedy wybierzesz jakiś typ przejdziesz na stronę, na której taka informacja się znajduje.

**Precyzja typu** wspomniana w ostatnim punkcie wiąże się z tym, że komputer nie potrafi zapisać dokładnie liczb z wartościami po przecinku. Może zapamiętać tylko przybliżenie. Ale czy przybliżeniem liczby 2.45678 będzie 2.5 czy 2.457 zależy właśnie od precyzji danego typu zmiennej. M.in. o tym mówi standard [**IEEE 754**](https://pl.wikipedia.org/wiki/IEEE_754).

Uzupełnijmy więc klasę naszego **konta rozliczeniowego** o potrzebne **pola** (**zmienne w klasie**):

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Bank

{

class BillingAccount

{

public string AccountNumber;

public decimal Balance;

public string FirstName;

public string LastName;

public long Pesel;

}

}

Co oznacza słowo public przed typem? O tym dowiesz się w późniejszym czasie. Na ten moment powiedzmy, że mówi ono, że do tak zaprojektowanej wnęki na dane będziemy mogli włożyć coś jako użytkownik wyprodukowanego z tego schematu obiektu.

Poza tym jednym słowem reszta wygląda podobnie jak wcześniej. O tym jak skorzystać z takich zmiennych dowiemy się w kolejnej lekcji.

**Ćwiczenie**

Dodaj w analogiczny sposób **klasę konta oszczędnościowego** (**SavingsAccount**). Skorzystaj z tych samych **typów danych**.

**Zmieniamy świat**

Dowiedzieliśmy się już jak dodawać w programie **pudełka na dane**, zwane **zmiennymi**.

A skoro są dane to i nasza aplikacja staje się coraz bardziej profesjonalna. Już tylko chwila dzieli nas od założenia pierwszego konta! Ruszajmy więc dalej.