**[KURS C#] Właściwe właściwości**



**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)!

Do tej pory do **przechowywania** i **modyfikowania** danych kont bankowych wykorzystywaliśmy **zmienne** i **metody**. I było nam z tym dobrze. Jednak po zapoznaniu się z **modyfikatorami dostępu** i **hermetyzacją** powinna Ci przyjść do głowy pewna myśl.

Skoro powinniśmy maksymalnie ograniczać dostęp do rzeczy znajdujących się w klasie to nie możemy tak po prostu oznaczać ich jako **publiczne**! Co prawda dzięki temu możemy odczytać wartość z jakiejś zmiennej, ale przecież nic nie stoi na przeszkodzie żeby w innym miejscu ktoś zmienił tą wartość. W końcu jest **publiczna**, prawda?

Prawda. Dlatego dobrym pomysłem będzie pewnie dodanie **publicznej metody**, która zwróci nam wartość zmiennej, ale już sama zmienna będzie **prywatna**. Ewentualnie będziemy mieli jakąś funkcję, która ustawi nam wartość tej zmiennej, ale w określony sposób. Świetny pomysł! W końcu funkcje w klasie mają dostęp do znajdujących się tam zmiennych niezależnie od tego czy są publiczne czy prywatne.

Funkcje, które służą do tego aby zwrócić wartość albo ją ustawić nazywamy zazwyczaj **getterami** i **setterami** (od angielskich **get** i **set**). Są one pomocne kiedy właśnie np. chcemy ograniczyć możliwość zmiany wartości jakiejś zmiennej, ale jednocześnie dać możliwość używania jej wartości na zewnątrz klasy. Ewentualnie jeżeli chcemy jakoś modyfikować wartość przed zwróceniem albo ustawieniem nowej.

Na szczęście twórcy języka **C#** pomyśleli, że jest to często wykorzystywany pomysł i udostępnili bardzo pomocną „nakładkę”. Nazywa się ona **właściwości**.

**Właściwie co to?**

Właściwości są jednym z tych elementów, które sprawiają, że programowanie jest dużo przyjemniejsze bo obudowują często wykorzystywaną strukturę w coś prostszego. W tym wypadku mamy połączenie **prywatnej zmiennej** i **publicznej lub prywatnej metody** dającej dostęp do tej zmiennej.

Prosty przykład:

class Test

{

**private** string **privateValue**;

**private** setValue(string value)

{

**privateValue** = value;

}

**public** getValue()

{

return privateValue;

}

}

Chcemy aby jakaś **wartość** w klasie mogła być pobrana w innym miejscu. Ale jednocześnie nie mamy powodu żeby dawać możliwość jej zmiany poza tą klasą. Dlatego w powyższym kodzie samą zmienną robimy **prywatną**, żeby przypadkiem nikt się do niej bezpośrednio nie dobrał. Funkcję, która ustawia tą wartość również robimy prywatną – będzie dostępna tylko wewnątrz klasy, np. w **konstruktorze**. Metoda do pobrania tej wartości jest jednak **publiczna** bo nie mamy problemu z tym, żeby ktoś ją odczytywał i wykorzystywał do swoich niecnych celów.

A co jeśli Ci powiem, że taki sam efekt osiągniemy zapisując wszystko w ten sposób:

class BetterTest

{

public string Value { get; private set; }

}

A nawet lepiej,  bo w nowszych wersjach języka **C#** jeżeli **get** albo **set** jest **prywatny** to nie musimy go pisać:

class BetterTest

{

public string Value { get; }

}

Tak oto wyglądają **właściwości**. W ten sposób załatwiliśmy sprawę **prywatnej zmiennej**, **publicznego gettera** i **prywatnego settera**.

Trzeba jedynie pamiętać, że ta druga postać ma pewne ograniczenie. Chodzi o to, że nie wpisując w ogóle **seta** możemy ustawić wartość takiej zmiennej **jedynie w konstruktorze**. Jeżeli chcielibyśmy ją zmieniać w innych funkcjach klasy to musimy zastosować pierwszą postać z jawnie podanym **prywatnym setterem**.

**Zabezpieczamy bank**

Z tą wiedzą wróćmy więc do naszej aplikacji bankowej. Mamy klasę **Account**, która odpowiada za przechowywanie informacji o kontach. A co się w niej dzieje?

namespace Bank

{

abstract class Account

{

public int Id;

public string AccountNumber;

public decimal Balance;

public string FirstName;

public string LastName;

public long Pesel;

public Account(int id, string firstName, string lastName, long pesel)

{

Id = id;

AccountNumber = generateAccountNumber(id);

Balance = 0.0M;

FirstName = firstName;

LastName = lastName;

Pesel = pesel;

}

}

}

Wszystkie dane **publiczne**! Ok, potrzebujemy móc z nich korzystać. Jednak przez to nie ma problemu, żeby w dowolnym miejscu w aplikacji je zmodyfikować w obiekcie tej klasy!

Wyobrażasz sobie, że ktokolwiek w banku może od tak wziąć i zmienić Twoje dane np. w przelewie? Albo zmienić stan Twojego konta po prostu wpisując nowy? Absolutnie nie!

Pierwszym rozwiązaniem jakie byśmy jeszcze do niedawna zastosowali to była by **publiczna metoda** do pobierania tych wartości, a one same były by **prywatne**. Jednak uzbrojeni w zdobytą przed chwilą wiedzę możemy ułatwić sobie życie i zastosować **właściwości** zamiast samych **zmiennych**.

Zamiast:

public int Id;

Wpisujemy:

public int Id { get; }

I tak dla wszystkich pól w klasie. Ostatecznie będzie ona wyglądać tak jak poniżej (usunąłem metody dla poprawienia czytelności):

namespace Bank

{

abstract class Account

{

public int Id { get; }

public string AccountNumber { get; }

public decimal Balance { get; }

public string FirstName { get; }

public string LastName { get; }

public long Pesel { get; }

public Account(int id, string firstName, string lastName, long pesel)

{

Id = id;

AccountNumber = generateAccountNumber(id);

Balance = 0.0M;

FirstName = firstName;

LastName = lastName;

Pesel = pesel;

}

}

}

**Co zyskaliśmy?**

Dzięki tej prostej zmianie **bez pisania dodatkowych metod** dla każdej wartości zapewniliśmy, że nadal będzie można odczytać tą wartość poza klasą. Jednocześnie jakakolwiek modyfikacja będzie możliwa tylko wewnątrz tej klasy.

Pod spodem nadal jest dodawana **zmienna** i **funkcja**. Jednak zastosowanie **właściwości** pozwoliło skrócić zapis i osiągnąć ten sam efekt jednak z zachowaniem dużo większej czytelności.

**A coś więcej?**

Jednak to co rozbiliśmy powyżej to tylko część możliwości właściwości. Chociaż nie da się ukryć, że najczęściej wykorzystywana.

**Coś dostępne, coś ukryte**

Kolejna opcja to oczywiście dodanie innych **modyfikatorów dostępu**. Ważne żeby zapamiętać, że **getter** i **setter**, jeżeli nie podamy przy nich modyfikatorów dostępu to przyjmują taki modyfikator jak ma **cała właściwość**.

Czyli jeżeli nasza właściwość jest **publiczna** to **get** bez dodatkowego modyfikatora też jest **publiczny**.

Nie ma też problemu, żeby setter był publiczny:

public int Id { get; set; }

Albo możemy mieć całą właściwość chronioną:

protected int Id { get; set; }

Albo właściwość i getter chronione ale setter dostępny tylko w konstruktorze:

protected int Id { get; }

**Mała zmiana**

Inną możliwością jest faktyczne skorzystanie z faktu, że **getter** i **setter** są **metodami**.  
Bo to, że przedstawiony powyżej sposób jest krótki i tylko ustawia lub pobiera wartość to wiemy. Ale możemy też dopisać jakąś **logikę** we **właściwościach**.  
Jednak kiedy to robimy to tracimy to, że automatycznie dodaje nam się prywatna zmienna pod spodem.  
Także jeżeli chcielibyśmy aby pobierana wartość była dwukrotnie większa niż faktycznie zapisana to możemy to zrobić w ten sposób:

private int someValue;

public int SomeValue

{

get

{

return someValue \* 2;

}

}

Jak widać musieliśmy tutaj dodać sami zmienną, która przechowuje wartość. Podobnie możemy zrobić dodając własny setter:

private int someValue;

public int SomeValue

{

get

{

return someValue;

}

set

{

someValue = **value** \* 2;

}

}

Wtedy korzystamy ze słowa kluczowego **value**, które reprezentuje wartość jaką próbujemy przypisać do **właściwości**.

Jeżeli dopisujemy jakiś kod w którejś z metod właściwości to niestety w drugiej z nich również musimy jawnie napisać zwracanie albo ustawianie wartości zmiennej. Jednak ten sposób wykorzystania właściwości jest dużo rzadziej używany, co nie znaczy, że nie warto go poznać.

W takim przypadku **właściwości** są jedynie skróconym zapisem funkcji getSomething() i setSomething().

**Właściwe podejście**

Jak widzisz korzystanie z **właściwości** pozwala w prostszy sposób kontrolować dostęp do **wartości** w klasie. Dzięki temu zapewniamy sobie w prostszy sposób **hermetyzację klasy**.

Od teraz prawdopodobnie dużo częściej będziesz korzystał z właściwości zamiast ze zmiennych w klasach. Jednak absolutnie nie jest to jednoznacznie najlepsze podejście bo oba mają swoje wady i zalety i nie zawsze potrzebujemy stosować właściwości, zwłaszcza jeżeli mówimy o prywatnych wartościach.