Właściwości (ang. property)

Do tej pory składnia naszego kodu wyglądała następująco:

```
class Osoba
{
    private string imie;
    private int wiek;

    public void UstawWiek(int wiek) { this.wiek = wiek; }
    public int PobierzWiek() { return wiek; }

    public void UstawImie(string imie) { this.imie = imie; }
    public string PobierzImie() { return imie; }
}
```

W metodzie Main klasy Program wyglądało to tak:

```
Osoba osoba1 = new Osoba();
osoba1.UstawWiek(21); //długo
```

A nie było dostępu do pól bezpośrednio z uwagi na hermetyzację:

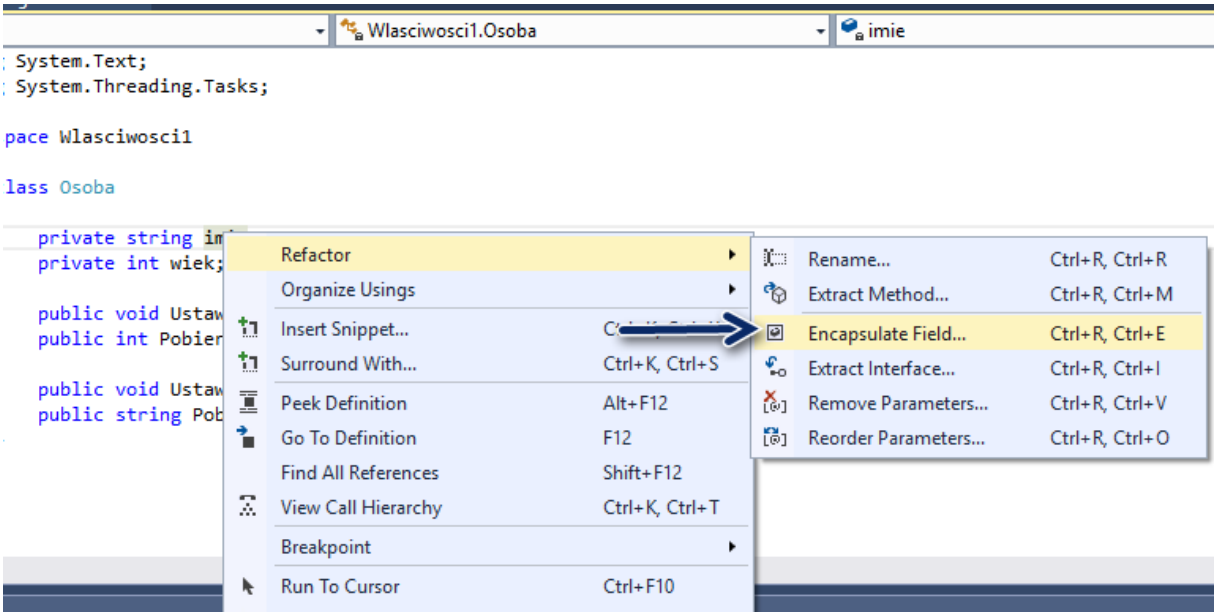
```
osoba1.wiek = 21; //niedostępne
```

Pełny kod: https://github.com/Piotrekk16/Wlasciwosci1.

Czym są zatem właściwości?

- specjalna konstrukcja języka C# pozwalająca połączyć zachowanie pól i metod
- wyglądają jak pola więc możemy pobrać ich wartość bądź je przypisać, a jednocześnie zachowują się jak metody czyli możemy stosować je w interfejsach

Z poziomu Visual Studio najłatwiej stworzyć prywatne pole i wybrać opcję z Menu (lub z prawego klawisza myszy):



Druga opcja to wpisanie propfull i naciśnięcie Tab:

```
private int myVar;

public int MyProperty
{
    get { return myVar; }
    set { myVar = value; }
}
```

Składnia właściwości:

```
private int wiek; //pole
public int Wiek //właściwość
{
    get { return wiek; }
    set { wiek = value; }
}
```

Zwróć uwagę na dwie rzeczy:

- nazwa pola (z małej litery) a nazwa właściwości (z dużej)

- value – występuję tylko we wnętrzu set

Użycie właściwości dla obiektu:

```
Osoba osoba1 = new Osoba();
osoba1.Wiek = 21;  //set
Console.WriteLine(osoba1.Wiek); //get
```

Pełny kod: https://github.com/Piotrekk85/Wlasciwosci2.

Skrócona deklaracja:

```
class Osoba
{
    public int Wiek { get; set; }
}
```

Uwaga: przy skróconej deklaracji niejawnie tworzone jest prywatne pole Wiek (widać do w trybie Debug):

Autos

Name	Value	Type
◢ osoba1	{ConsoleApplication20.	ConsoleApplication20.Osoba
Wiek	21	int
osoba1.Wiek	21	int

Właściwość tylko do odczytu:

```
private int wiek;

public int Wiek
{
    get
    {
        return wiek;
    }
}
```

Właściwość tylko do zapisu:

```
private int wiek;

public int Wiek
{
    set
    {
        wiek = value;
    }
}
```

Dobra praktyka wg MSDN:

- ilość kodu/instrukcji umieszczonych we właściwościach powinniśmy ograniczyć do minimum
- dopuszczalne są jedynie „proste” operacje

Dostępność właściwości:

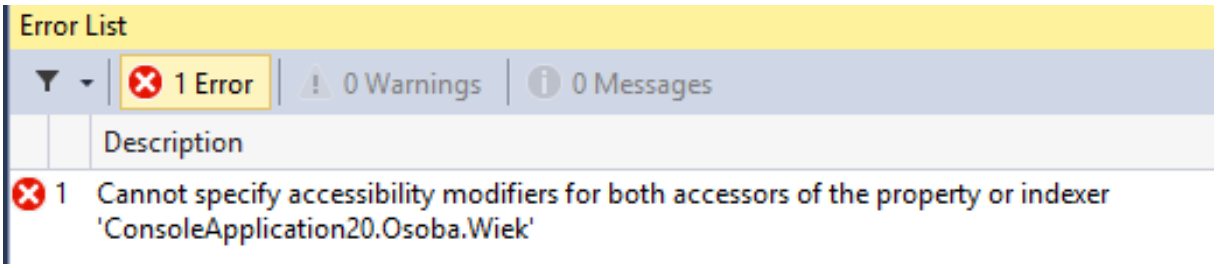
- domyślne powielany jest modyfikator dostępu właściwości dla `get` i `set`, ale można go nadpisać

```
public int Wiek
{
    private get { return wiek; }
    set { wiek = value; }
}
```

- nie możemy nadpisać obu jednocześnie (lepiej zmienić modyfikator właściwości)

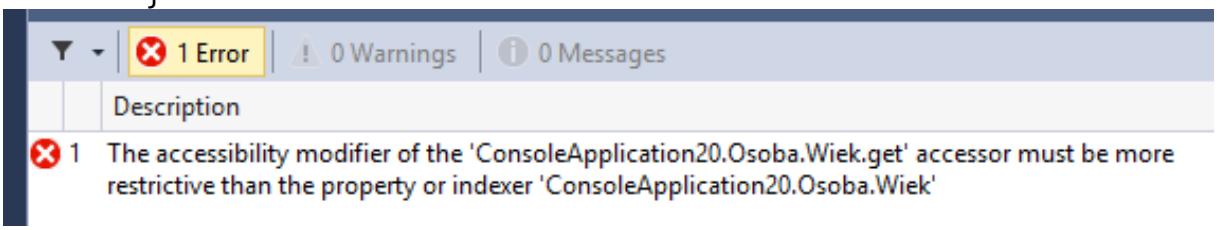
```
public int Wiek
{
    private get { return wiek; }
    private set { wiek = value;
}
```

Powyższy kod skutkuje błędem kompilacji:



- modyfikatory dla `get` i `set` nie mogą być bardziej dostępne niż modyfikator samej właściwości

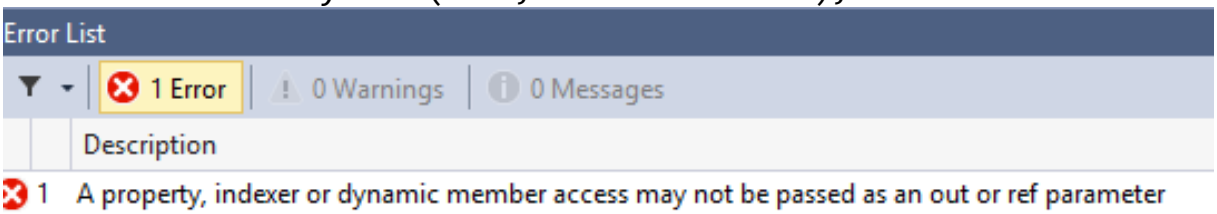
```
private int Wiek
{
    public get { return wiek; }
    set { wiek = value; }
}
```



Pola czy właściwości – co używać?

- właściwości nie mogą być użyte przy słowach kluczowych `ref` i `out`

```
Osoba osoba1 = new Osoba();
int.TryParse("21", out osoba1.Wiek);
```



- właściwości nie mogę być stałymi

Interfejsy a właściwości

- jako że właściwości są metodami, możemy je wrzucać do interfejsów

```
interface IWiek
{
    int Wiek { get; set; }
}
```

Właściwości wirtualne i ich przesłanianie

- tak jak do innych metod możemy użyć słów kluczowych `virtual`, `override` i `abstract`, na przykład:

```
class Osoba
{
    protected int wiek;
    public virtual int Wiek
    {
        get { return wiek; }
        set { wiek = value; }
    }
}
class Dwudziestolatek : Osoba
{
    public override int Wiek
    {
        get { return 20; }
        set { wiek = value; }
    }
}
```

Inicjowanie obiektu z użyciem właściwości

```
Osoba osoba1 = new Osoba() { Wiek = 22 };
```