Table of Contents

[Wstęp Do Programowania Obiektowego: 2](#_Toc63000570)

[Klasa A Obiekt: 2](#_Toc63000571)

[Istota Programowania Obiektowego: 2](#_Toc63000572)

[Tworzenie Klasy: 3](#_Toc63000573)

[Metody, Atrybuty I Konstruktory Klas: 8](#_Toc63000574)

[Metoda Klasy (Czym Jest I Do Czego Służy): 8](#_Toc63000575)

[Deklarowanie Metody: 9](#_Toc63000576)

[Atrybuty Klasy: 12](#_Toc63000577)

[Definiowanie Atrybutów: 13](#_Toc63000578)

[Konstruktory: 13](#_Toc63000579)

[Deklaracja Konstruktora: 15](#_Toc63000580)

[Przestrzenie Nazw, Modyfikatory Dostępu: 16](#_Toc63000581)

[Istota Przestrzeni Nazw: 16](#_Toc63000582)

[Modyfikatory Dostępu: 16](#_Toc63000583)

[Dostępne Modyfikatory Dostępu: 17](#_Toc63000584)

[Wstęp Do Dziedziczenia: 19](#_Toc63000585)

[przykład Dziedziczenia 20](#_Toc63000586)

[Ilustracja Mechanizmu Dziedziczenia 22](#_Toc63000587)

[Polimorfizm: 23](#_Toc63000588)

[Przykład Zastosowania Mechanizmu Polimorfizmu: 24](#_Toc63000589)

# Wstęp Do Programowania Obiektowego:

## Klasa A Obiekt:

Klasa ­– Opis nowej struktury danych

Obiekt – Struktura danych stworzona zgodnie z opisem klasy

## Istota Programowania Obiektowego:

Programowanie obiektowe umożliwia opisanie cech oraz funkcjonalności elementów, na przykład drukarki, monitora, klawiatury, samochodu itd. za pomocą abstrakcyjnego pojęcia zwanego klasą, które ułatwia implementację danego problemu.

Dzięki podejściu obiektowemu możliwy jest logiczny podział projektu na komponenty (obiekty), w celu ułatwienia implementacji rozwiązania a w szczególności ułatwienia rozbudowy projektu.

## Tworzenie Klasy:

W celu utworzenia klasy należy umieścić poniższy kod, w tym przypadku będzie to klasa BasicUser:



*Rysunek 1 utworzenie klasy*

W celu utworzenia obiektu klasy BasicUser należy umieścić poniższy kod:

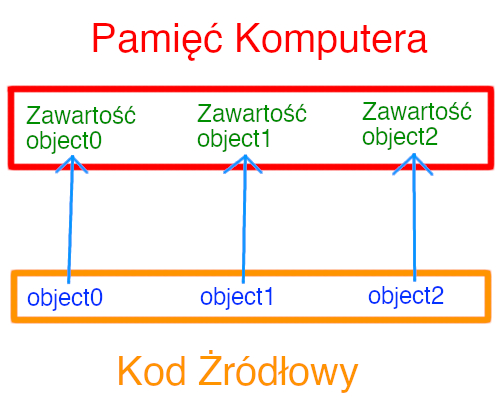


*Rysunek 2 utworzenie Obiektu Klasy*

Obiekt klasy (typu) BasicUser jest tworzony i umieszczany w zasięgu metody Main.

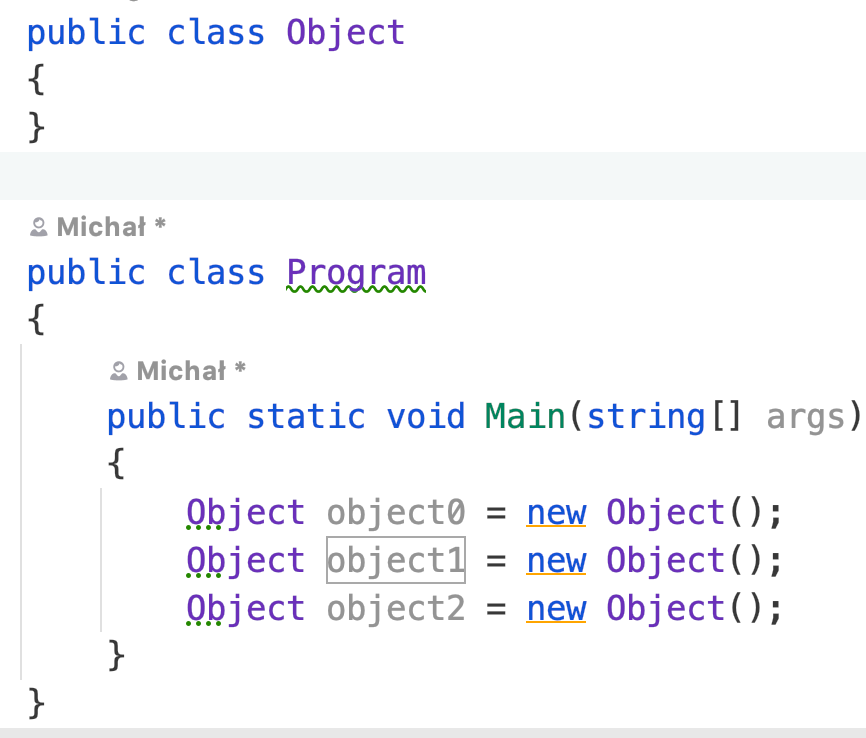
Słowo kluczowe **namespace** będzie omawianie w następnych rozdziałach, tyczy się to także modyfikatora dostępu **public**, słowa kluczowego **static** oraz wyjaśnienia pojęcia **metod** w programowaniu obiektowym.

Mechanizm Generowania Obiektów Klas:



*Rysunek 3 kod źródłowy a pamięć komputera*

Możliwe jest utworzenie kilku obiektów tej samej klasy. Istotny jest fakt, że każdy obiekt jest niezależny względem pozostałych obiektów, które powstały i tych, które mogą później zostać zdefiniowane.



*Rysunek 4 kod źródłowy Do rysunku3*

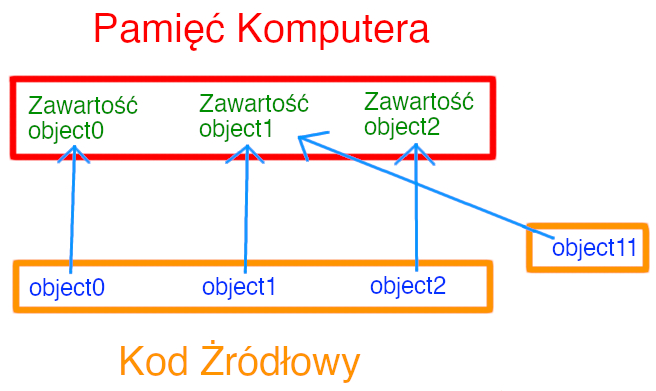
W procesie definiowania nowych obiektów ważnym aspektem jest umieszczenie słowa kluczowego **new**, które sprawia, że tworzona jest nowa, niezależna instancja obiektu określonej klasy.

Odwołując się do rysunku 3 można zauważyć, że każdy element zdefiniowany w kodzie jako object0, object1, object2 jest w swej istocie jedynie aliasem (referencją) obiektu, który jest umieszczony w określonym miejscu pamięci komputera.

Biorąc ten fakt pod uwagę poniższy kod powoduje, że istnieją dwa odniesienia do zawartości obiektu object1, co daje możliwość operowania na zawartości obiektu1 za pomocą dwóch aliasów, w tym przypadku object11.



*Rysunek 5 kod źródłowy z dodatkową referencją*



*Rysunek 6 Uwzględnienie nowej referencji*

# Metody, Atrybuty I Konstruktory Klas:

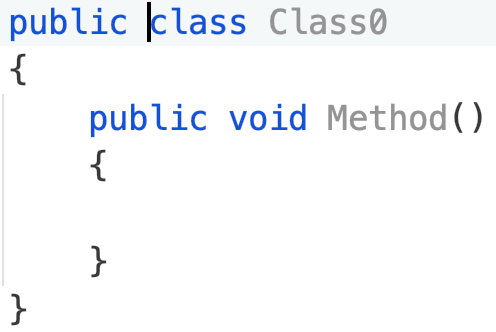
## Metoda Klasy (Czym Jest I Do Czego Służy):

Metoda jest odpowiednikiem funkcji w podejściu strukturalnym. Metoda wchodzi w skład klasy, umożliwiając operowanie na danych będących:

* Elementami składowymi klasy
* Danymi, przekazywanymi do metody za pomocą argumentów formalnych
* Danymi zaalokowanymi statycznie (dostępnych w obrębie całej aplikacji, bądź określonej przestrzeni nazw)
* Danymi zaalokowanymi w zasięgu bloku metody

## Deklarowanie Metody:

W celu deklaracji należy stworzyć klasę i zdefiniować metodę, tak jak na przykład ilustruje to poniższy kod:

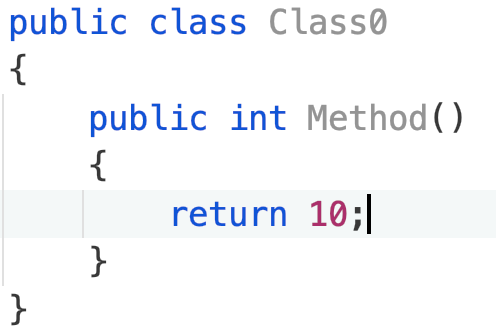


*Rysunek 7 Deklaracja Metody typu void*

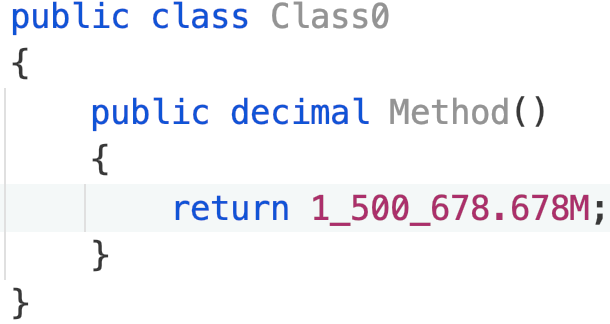
Słowo kluczowe **public** zostanie omówione w następnym rozdziale.

Istotne jest w tym przypadku słowo kluczowe **void**, które informuje, że metoda będzie przetwarzać dane, nie zwracając żadnej wartości po zakończeniu jej wykonywania w miejscu jej wywołania.

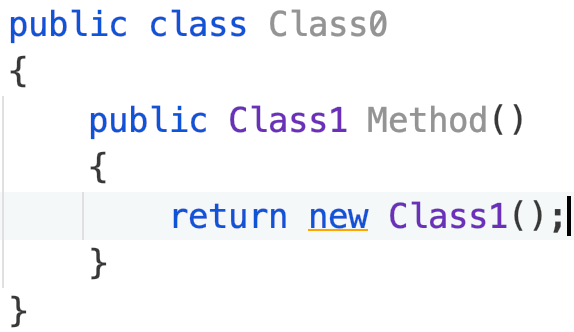
Deklaracja każdej metody może być opatrzona słowem kluczowym **void** bądź **typem prostym** bądź **typem obiektu**. W dwóch ostatnich przypadkach oznacza to, że po zakończeniu wykonywania metody, zwróci ona dane typu zdefiniowanego w deklaracji metody.



*RYSUNEK 8 DEKLARACJA METODY TYPU prostego (int)*



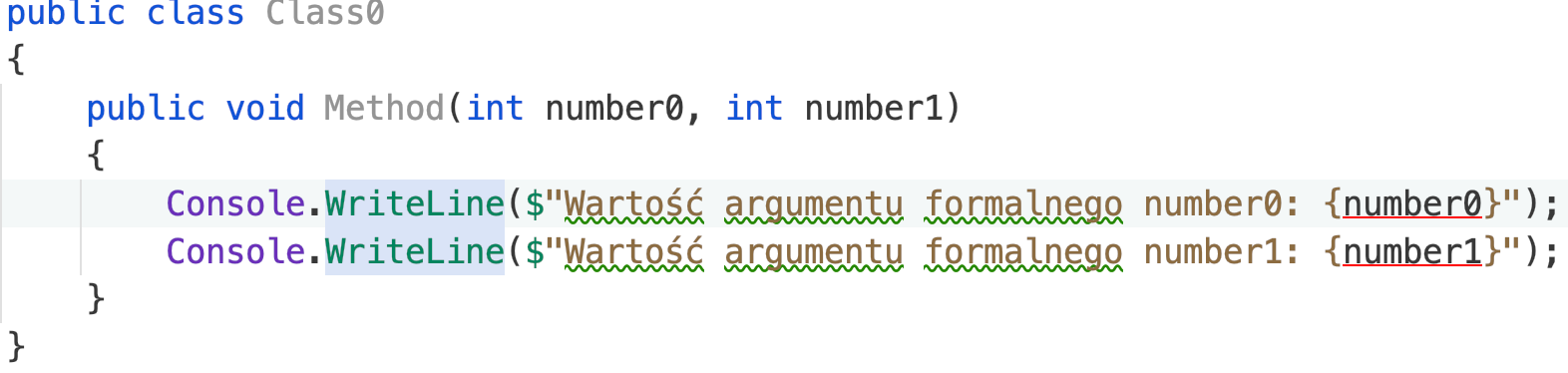
*RYSUNEK 9 deklaracja metody typu prostego (decimal)*



*Rysunek 10 DEKLARACJA METODY TYPU obiekt*

Każda metoda zwracająca wartość (**typ prosty**, bądź **typ obiektu**) musi posiadać instrukcję **return**, informującą jaka wartość, bądź obiekt będzie zwrócona/zwrócony.

Jak wcześniej opisano, metoda może przyjmować wartości zwane argumentami formalnymi. W celu zdefiniowania takiej metody można posłużyć się poniższym kodem:



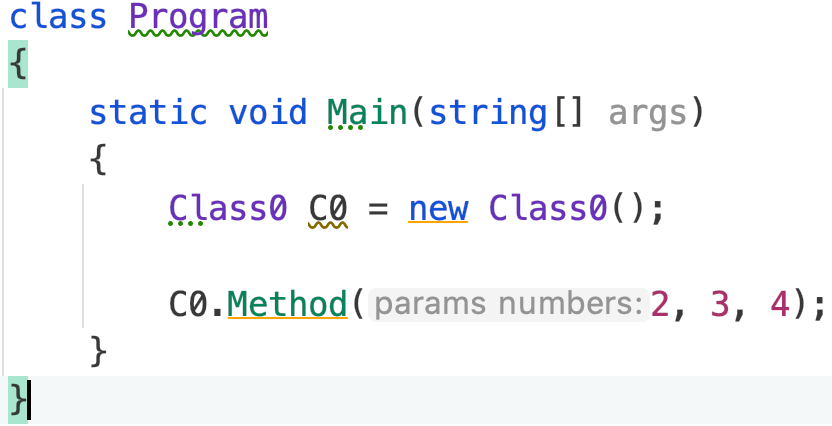
*RYSUNEK 11 DEKLARACJA METODY z parametrami formalnymi*

W celu wyświetlenia przekazanych wartości do metody w terminalu użyto metody **WriteLine** klasy **Console** i **interpolacji łańcuchów znakowych** ($”{} {} {}”).

Możliwe jest zdefiniowanie metody przyjmującej dowolną ilość danych danego typu. W celu zdefiniowania takiej metody należy skorzystać ze słowa kluczowego **params.**

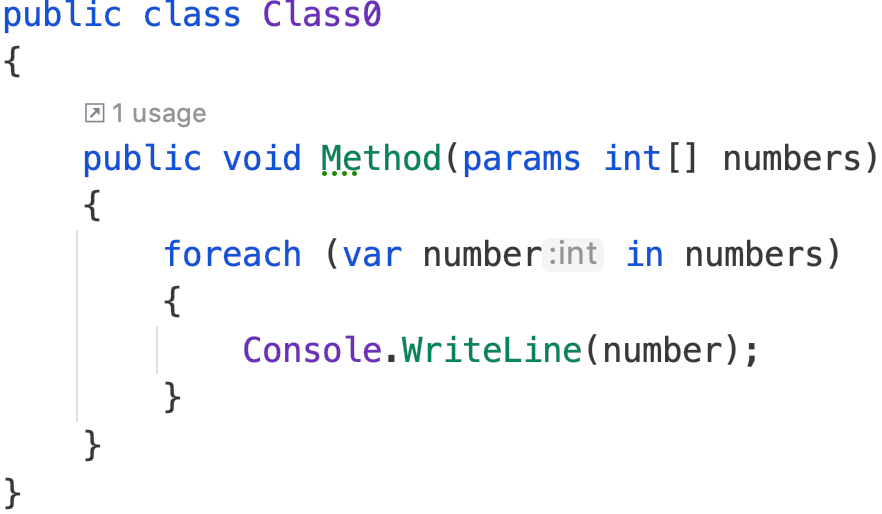
Przykładowa definicja metody korzystającej z wyżej opisywanego mechanizmu w celu wyświetlenia wszystkich przesłanych danych typu int została zilustrowana poniżej:

## Atrybuty Klasy:



*RYSUNEK 13 Przykładowe wykorzystanie metody ze zmienną ilością parametrów*

*RYSUNEK 12 DEKLARACJA METODY ze zmienną  
 ilością parametrów*

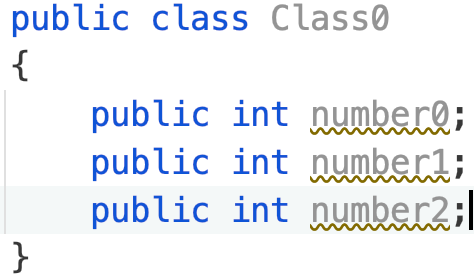


Atrybuty są elementami składowymi klas służącymi do przechowywania danych związanych z daną klasą. Dane te mogą być:

* Typem prostym (int, long, double, decimal itd.)
* Typem złożonym (obiekt, struktura danych itp.)

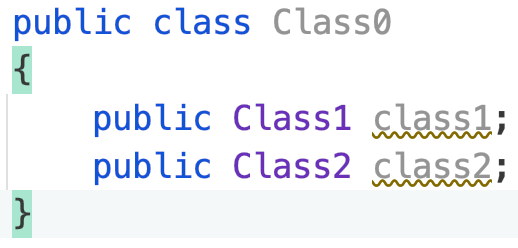
## Definiowanie Atrybutów:

W celu zdefiniowania atrybutów klasy można posłużyć się poniższym przykładem:



*RYSUNEK 14 DEKLARACJA atrybutów klasy*

## Konstruktory:



*RYSUNEK 15 DEKLARACJA atrybutów klasy jako obiektów*

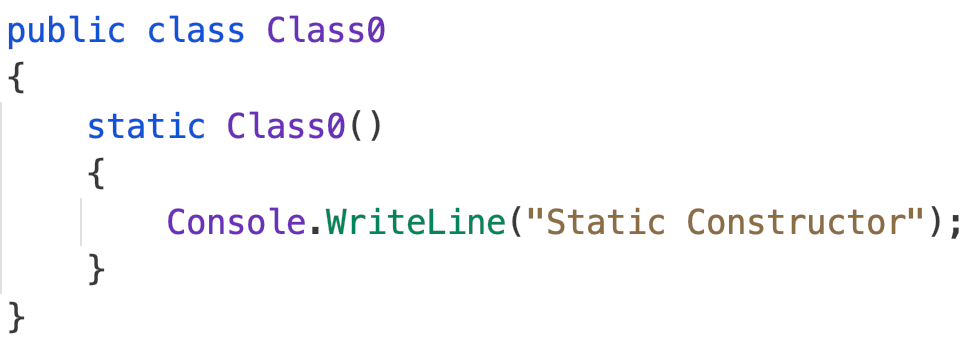
Konstruktor jest to metoda, wywoływana w chwili utworzenia nowej instancji klasy. Konstruktor może być:

* Bezparametryczny (sygnatura konstruktora jest pusta)
* Wieloargumentowy (sygnatura konstruktora składa się   
  z przynajmniej jednego argumentu formalnego)
* Statyczny (Wywoływany w momencie pierwszej interakcji z klasą, na przykład utworzenie nowej instancji klasy, bądź odczytanie wartości statycznego atrybutu klasy)

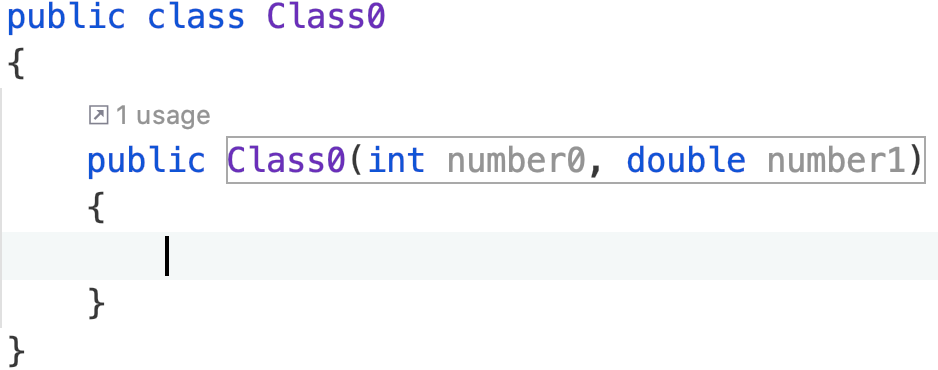
## Deklaracja Konstruktora:



*RYSUNEK 16 DEKLARACJA Konstruktora Bezparametrycznego*



*RYSUNEK 18 DEKLARACJA Konstruktora statycznego*



*RYSUNEK 17 DEKLARACJA Konstruktora wieloargumentowego*

Blok konstruktora może posiadać instrukcje, które mają być wykonane w trakcie obsługi kodu konstruktora, jak na przykład: wyświetlenie określonego łańcucha znakowego.

# Przestrzenie Nazw, Modyfikatory Dostępu:

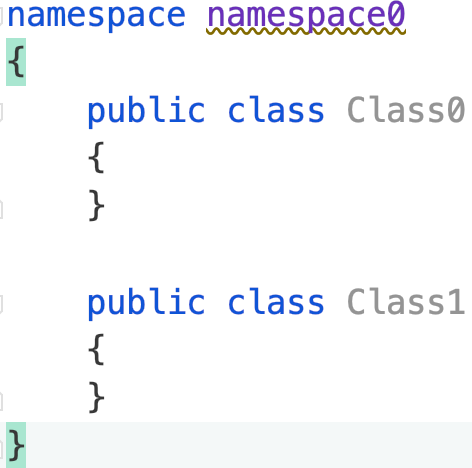
## Istota Przestrzeni Nazw:

Przestrzeń nazw jest abstrakcyjnym obszarem, służącym do grupowania elementów składowych projektu, jak na przykład:

* Pojedyncze Klasy
* Biblioteki DLL (Dynamic Link Library)
* Programy Wykonywalne

W celu zdefiniowania przestrzeni nazw i zawartych w niej elementów należy użyć słowa kluczowego **namespace**, co ilustruje poniższy kod:

## Modyfikatory Dostępu:



*RYSUNEK 19 DEKLARACJA przestrzeni nazw*

Modyfikatory dostępu **służą do określenia dostępu** do elementów składowych klasy. Ograniczanie dostępu do elementów składowych stosuje się w celu hermetyzacji danych, która jest jedną z głównych zasad programowania obiektowego.

### Dostępne Modyfikatory Dostępu:

public – Dostęp do elementów składowych jest nieograniczony

Dostęp Do Zasobu Mają:

* Metody składowe klasy
* Klasy dziedziczące po klasie
* Klasy korzystające z instancji klasy

protected – Dostęp do elementów składowych jest ograniczony

Dostęp Do Zasobu Mają:

* Metody składowe klasy
* Klasy dziedziczące po klasie

private – Dostęp do elementów składowych jest ograniczony

Dostęp Do Zasobu Mają:

* Metody składowe klasy

internal – Dostęp do elementów składowych jest ograniczony

Dostęp Do Zasobu Mają:

* Metody składowe klasy
* Klasy wchodzące w skład tej samej przestrzeni nazw

protected internal – Dostęp do elementów składowych jest ograniczony

Dostęp Do Zasobu Mają:

* Metody składowe klasy
* Klasy dziedziczące po klasie
* Klasy wchodzące w skład tej samej przestrzeni nazw

private protected – Dostęp do elementów składowych jest ograniczony

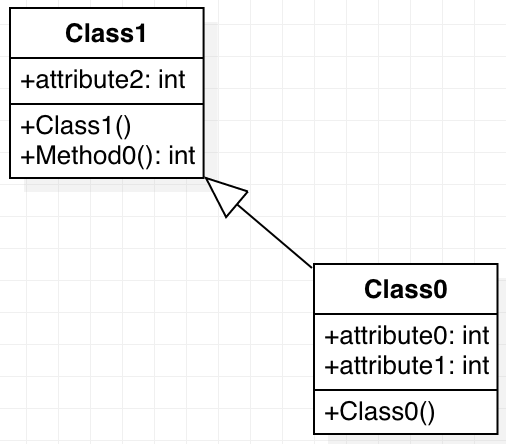
Dostęp Do Zasobu Mają:

* Metody składowe klasy
* Klasy dziedziczące po klasie

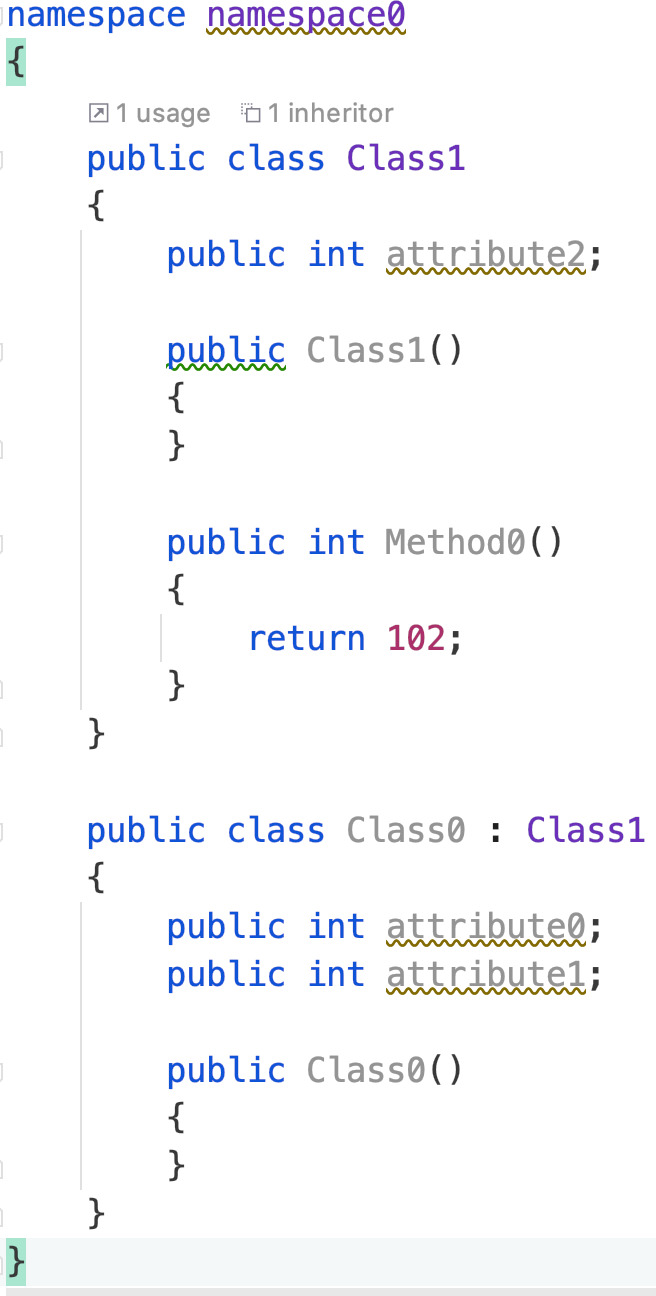
# Wstęp Do Dziedziczenia:

Dziedziczenie jest mechanizmem umożliwiającym **implementowanie cech klas do innych klas**. Dzięki temu, możliwe jest łatwiejsze modelowanie programu, co wiąże się z krótszym czasem projektowania schematu projektu.

## przykład Dziedziczenia



*RYSUNEK 20 diagram klas w języku uml*

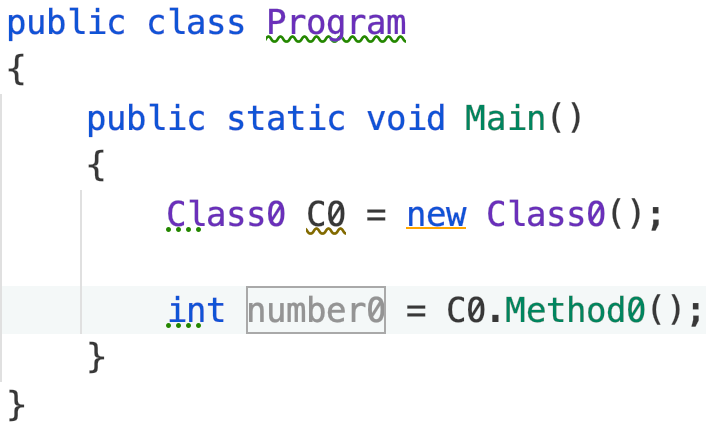


*RYSUNEK 21 odwzorowanie diagramu klas w kod źródłowy*

Powyższy kod źródłowy, pokazuje, że aby zastosować mechanizm dziedziczenia należy dopisać do nazwy klasy **‘: ,<nazwa klasy bazowej>’**. Zabieg taki powoduje, że klasa staje się **klasą pochodną** (Class0), dziedzicząc po **klasie bazowej** (Class1).

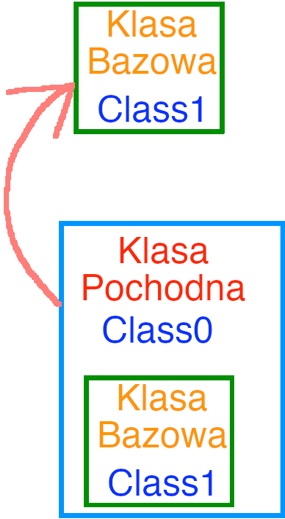
Każda klasa dziedzicząca po innej klasie ma dostęp do **elementów składowych** instancji takiej klasy opatrzonych klauzulą **public**, **protected**, **protected** **internal** bądź **private** **protected**.

Poniższy kod ilustruje przykład użycia elementów składowych **klasy bazowej** w instancji **klasy pochodnej**:



*RYSUNEK 22 Przykład zastosowania dziedziczenia*

## Ilustracja Mechanizmu Dziedziczenia



*RYSUNEK 23 Ilustracja Mechanizmu dziedziczenia*

Zgodnie z powyższym przykładem kodu źródłowego, w momencie tworzenia klasy **Class0** tworzonajest dodatkowo instancja klasy **Class1**, która jest **zagnieżdżona** w instancji klasy **Class0**, dzięki czemu możliwe jest odwoływanie się do jej atrybutów bądź metod.

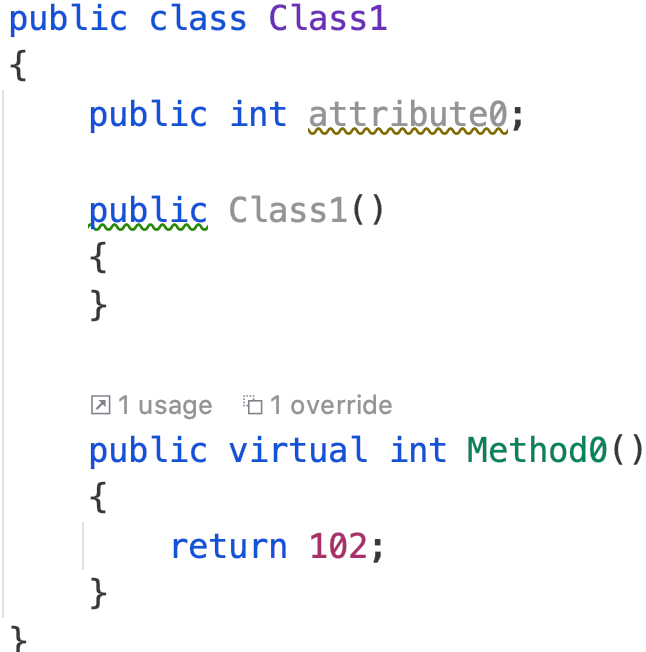
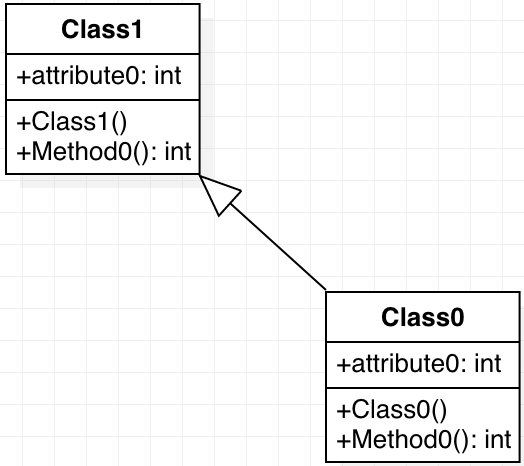
Przyswojenie tej właściwości może pomóc w zrozumieniu zagadnienia **polimorfizmu**, które będzie omawiane w kolejnych rozdziałach.

# Polimorfizm:

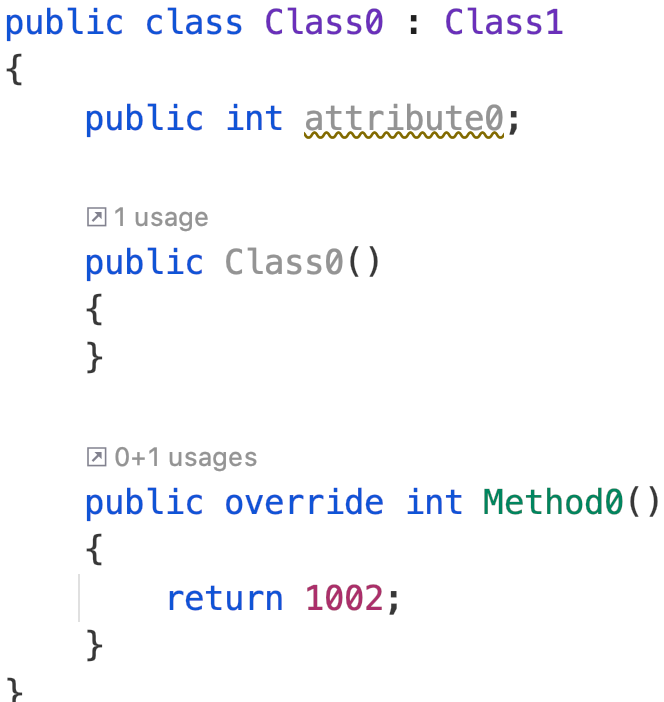
Polimorfizm jest mechanizmem pozwalającym na używanie atrybutów, metod itp. na kilka różnych sposobów, poprzez korzystanie z **referencji polimorficznej**.

## Przykład Zastosowania Mechanizmu Polimorfizmu:

*RYSUNEK 24 model klas do wykorzystania mechanizmu polimorfizmu*



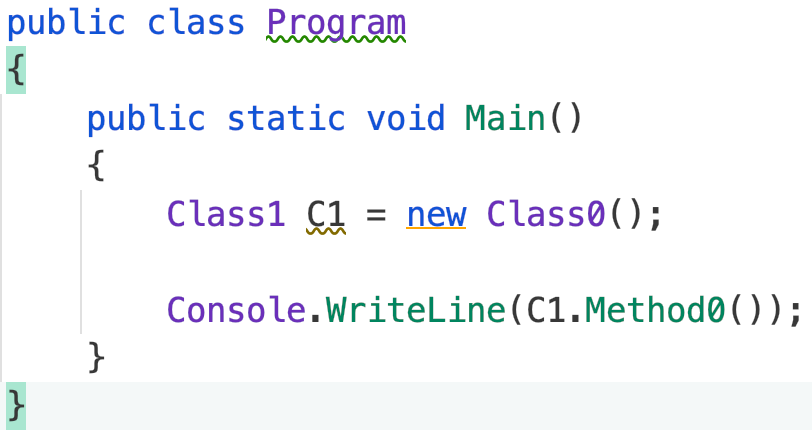
*RYSUNEK 25 Deklaracja klasy bazowej*



*RYSUNEK 26 Deklaracja klasy pochodnej*

W celu zdefiniowania klas korzystających z właściwości **polimorfizmu**, należy:

* W klasie bazowej użyć słowa kluczowego **virtual,** które sprawia, że metoda staje się **wirtualna** (wywołanie jej za pomocą **referencji polimorficznej** umożliwia korzystanie z definicji metody klasy, na którą wskazuje referencja)
* W klasie pochodnej użyć słowa kluczowego **override**, które sprawia, że definicja metody będzie uwzględniana w przypadku użycia **referencji polimorficznej**



*RYSUNEK 27 użycie klas obsługujących mechanizm polimorfizmu*

Zdefiniowanie referencji obiektu klasy bazowej   
(**referencja polimorficzna**) i przypisanie do niej obiekt klasy pochodnej z uprzednio zastosowanymi operacjami daje możliwość korzystania z elementów składowych klasy pochodnej za pomocą referencji klasy bazowej.

Dzięki takiemu rzutowaniu, możliwe jest ograniczenie obsługi atrybutów, metod itp. klasy pochodnej do elementów składowych zagnieżdżonej instancji klasy bazowej w obiekcie klasy pochodnej.

created by Michał Kopiel

……………………………