

Zawartość:

- enum w C#
- enum w C++
- enum vs enum class w C++

Enum – element języka, który służy do definiowania grupy stałych

Zacznijmy od C# bo w nim, będzie łatwiej pokazać zastosowanie:

```
5 references
public enum MoveDirection
{
    Left,
    Right,
    Up,
    Down
}
```

Prosty enum wyznaczający cztery stałe oznaczające kierunek. Napiszmy funkcję, która przesuwa pewien obiekt w stronę podaną przez parametr.

Podejście nr1:

```
0 references
static void MoveObjectI(object toMove, int moveDirection)
{
    if (moveDirection == 1) { }
    else if (moveDirection == 2) { }
    else if (moveDirection == 3) { }
    else if (moveDirection == 4) { }
    else { } //ponieważ możliwe jest podanie innej wartości niż te wyżej
}
```

Taka funkcja jest poprawna i zadziała. Ma jednak dwie wady: nie wiemy co oznacza każda z liczb (trzeba pewnie analizować kod żeby to wywnioskować) oraz możliwe jest podanie wartości nieobsługiwanej (wartości innej niż 1,2,3,4) no bo przecież to int.

Podejście nr2:

```
0 references
static void MoveObjectI(object toMove, string moveDirection)
{
    if (moveDirection == "left") { }
    else if (moveDirection == "right") { }
    else if (moveDirection == "up") { }
    else if (moveDirection == "down") { }
    else { } //ponieważ możliwe jest podanie innej wartości niż te wyżej
}
```

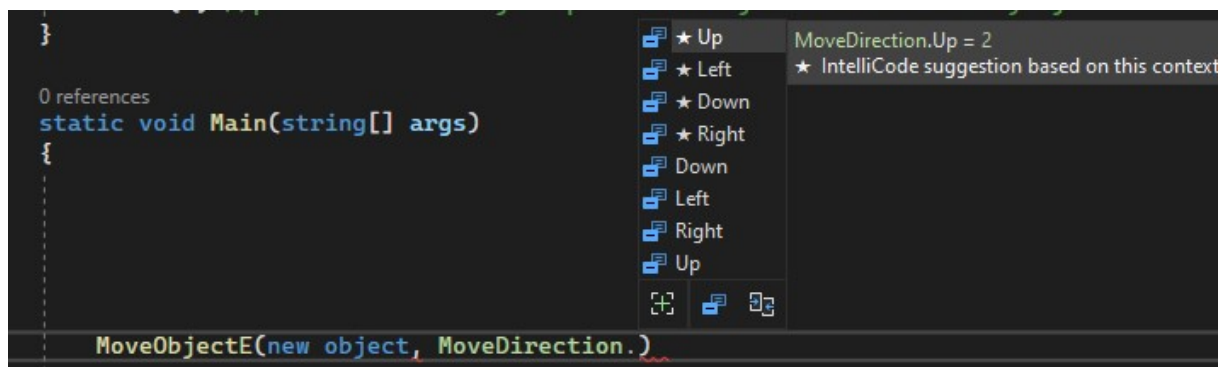
Fajnie. Rozwiązaliśmy pierwszy problem z poprzedniego przykładu (choć nie do końca bo użytkownik, który używa funkcji musi wiedzieć, że kierunek w górę to „up” a nie np. „top”). Drugi problem wciąż nie jest rozwiązany bo ktoś może podać „Uśmiechnij się !” jako parametr kierunku 😊

Podejście nr3:

```
0 references
static void MoveObjectE(object toMove, MoveDirection moveDirection)
{
    if (moveDirection == MoveDirection.Left) { }
    else if (moveDirection == MoveDirection.Right) { }
    else if (moveDirection == MoveDirection.Up) { }
    else if (moveDirection == MoveDirection.Down) { }
}
```

Wykorzystujemy enum. Teraz każdy kierunek ma określoną wartość z przyjazną nazwą. IDE podpowie dostępne opcje. Pierwszy problem z głowy.

Drugi problem także mamy z głowy. Enum przewiduje 4 wartości. IDE nie pozwoli wprowadzić innej, ponieważ skończy się to błędem kompilacji i ostrzeżeniem ze strony IDE.



The screenshot shows a code editor with a static void Main method. A code completion menu is open, listing the values of the MoveDirection enum: Up, Left, Down, Right, Down, Left, Right, Up. The first 'Up' is selected. To the right of the menu, a tooltip displays 'MoveDirection.Up = 2' and '★ IntelliCode suggestion based on this context'. Below the menu, the code 'MoveObjectE(new object, MoveDirection.)' is visible with a red squiggly line under the dot.

Jak widać w prawym górnym rogu, każdy element enum dostaje wartość liczbową w prezencie zgodnie z kolejnością definicji. Możemy także sami nadać wartość liczbową elementowi enum.

```
6 references
public enum MoveDirection
{
    Left = 2,
    Right = 11,
    Up = 9,
    Down = 12
}
```

W tym przypadku liczby te nie są powiązane z kierunkami, ale może nam się trafić taki enum gdzie warto będzie sparować liczby ze stałymi by wykorzystać je do obliczeń dalej:

```
Console.WriteLine(MoveDirection.Up.ToString()); //wypisanie jako string
Console.WriteLine((int)MoveDirection.Up); //wypisanie jako int
```

ToString() wypisze oczywiście „Up”

W języku C++ koncept jest bardzo podobny. Enum służy do definiowania grupy stałych.

W C++ istnieją dwie bardzo podobne konstrukcje:

- Enum
- Enum class

Nie są one identyczne.

```
enum MoveDirections{
    Left,
    Right,
    Up,
    Bottom,
}

enum ResizeDirection{
    Left,
    Right,
    Up,
    Bottom
}
```

Taki kod spowoduje błąd kompilacji, ponieważ zmienna Left (oraz pozostałe) zostały zdefiniowane dwa razy w programie. Zwykły enum nie separuje definicji w nim zawartych od reszty programu, dlatego mamy tutaj dubla każdej z czterech stałych.

```
enum class MoveDirections{
    Left,
    Right,
    Up,
    Bottom,
};

enum class ResizeDirection{
    Left,
    Right,
    Up,
    Bottom
};
```

Taki kod nie spowoduje błędu, ponieważ definicje są definiowane wewnątrz przestrzeni każdego z enum class. To pierwsza różnica.

Dla uproszczenia dalszych przykładów przyjmijmy oznaczenie, że enum klasowy zostanie oznaczony przyrostkiem EC.

```
enum MoveDirections{
    Left,
    Right,
    Up,
    Bottom,
};

enum ResizeDirections{
    RLeft,
    RRight,
    RUp,
    RBottom,
};

enum class MoveDirectionsEC{
    Left,
    Right,
    Up,
    Bottom,
};

enum class ResizeDirectionsEC{
    RLeft,
    RRight,
    RUp,
    RBottom,
};
```

Sprawdźmy kilka ciekawych przypadków:

```
//zwykły enum
MoveDirections a = MoveDirections::Left;
ResizeDirections b = ResizeDirections::RLeft;

if (a == b) std::cout<<"Equal"<<std::endl; //wynik to EQUAL
else std::cout<<"Not Equal"<<std::endl;
```

Dostaniemy tutaj ostrzeżenie, że porównujemy różne typy, ale wynik porównania i tak będzie true. Jest to bardzo zły przypadek, ponieważ może prowadzić do Bugów.

Dlaczego tak jest? Dlatego iż pod stałymi enum, siedzą ponownie inty. Obie stałe Left oraz RLeft mają pod spodem wartość 0. Jeżeli porównamy Left z RRight to wynik oczywiście da fałsz. To samo jeżeli do każdej stałej przypiszemy innego inta.

```
//klasowy enum
MoveDirectionsEC a = MoveDirectionsEC::Left;
ResizeDirectionsEC b = ResizeDirectionsEC::RLeft;

if (a == b) std::cout<<"Equal"<<std::endl; //wynik to EQUAL
else std::cout<<"Not Equal"<<std::endl;

std::cout<<(int)a<<std::endl;
std::cout<<(int)b<<std::endl;
```

Ten kod bazuje na enumach klasowych. Porównanie takich stałych nie jest możliwe i zakończy się błędem kompilacji. Dlatego też enum klasowy jest bezpieczniejszy.

Wróćmy do przykładów z C# i przenieśmy je na C++:

```
void moveObject(Object& object, std::string moveDirection){
    if (moveDirection == "left") {}
    else if (moveDirection == "right") {}
    else if (moveDirection == "up") {}
    else if (moveDirection == "down") {}
    else {}
}
```

Wersja bazująca na stringach działa tak samo jak w C#. I ma te same problemy.

```
void moveObject(Object& object, int moveDirection){
    if (moveDirection == 1) {}
    else if (moveDirection == 2) {}
    else if (moveDirection == 3) {}
    else if (moveDirection == 4) {}
    else {}
}
```

Wersja z intami działa tak samo jak w C#. I ma te same problemy. Chociaż...

```
#define UP 1
#define DOWN 2
#define LEFT 3
#define RIGHT 4

void moveObject(Object& object, int moveDirection){
    if (moveDirection == LEFT) {}
    else if (moveDirection == RIGHT) {}
    else if (moveDirection == UP) {}
    else if (moveDirection == DOWN) {}
    else {}
}
```

Dzięki makrom preprocesora nazwaliśmy stałe intowe dzięki czemu łatwiej zorientować się, która co oznacza. Jednak nie eliminuje to problemu podania wartości np. 88 w parametrze. Takie rozwiązanie jest całkiem ok.

```
void moveObject(Object& object, MoveDirections moveDirection){  
    if (moveDirection == MoveDirections::Left) {}  
    else if (moveDirection == MoveDirections::Right) {}  
    else if (moveDirection == MoveDirections::Up) {}  
    else if (moveDirection == MoveDirections::Bottom) {}  
}
```

Wersja z enum.