

ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej

PWr

Spotkanie 6

*Aplikacje webowe na platformie .NET*Laboratorium – **Lista 6**

Wstęp.

Celem listy jest zapoznanie się z działaniem wybranych elementów języka C#.

Niejawne typy danych.

Język C#, podobnie jak wiele języków tej grupy (Java, C++ itd.) wymaga podania typu zmiennej podczas jej deklaracji. Składnia jest taka sama jak dla wspomnianych języków, czyli:

```
<variableType> <variableName>,
```

np.

```
double number
```

Podczas kompilacji ustalany jest adres (referencja) takiej zmiennej i rezerwuje się dla niej odpowiednią liczbę jednostek pamięci (bajtów). Typ zmiennej nie ulega zmianie do końca jej używania. Jednak oprócz deklaracji zmiennej można też od razu zainicjować ją początkową wartością np.:

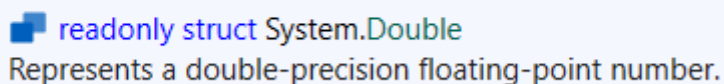
```
double number=1.23
```

W tym drugim przypadku można użyć niejawnego typu danych. Zamiast podawać konkretny typ zmiennej, wystarczy słowo kluczowe **var**, np.:

```
var number=1.23
```

Nie oznacza to, że zmienna jest nieznanego typu, czy ten typ może się zmienić. Typ zmiennej jest po prostu określany przez kompilator na podstawie typu wyniku wyrażenia po prawej stronie znaku równości. I typ ten już się nie zmieni. W wielu środowiskach programistycznych po ustawieniu wskaźnika myszki nad słowem **var** otrzymamy informację jaki typ zmiennej został ustalony przez kompilator np.:

```
var number = 1.23;
```




readonly struct System.Double
Represents a double-precision floating-point number.

Czyli w tym przypadku uznał ten typ za `System.Double`.

Podobnie umieszczenie wskaźnika myszki nad identyfikatorem zmiennej otrzymamy jej typ:

```
var number = 1.23;
```



(zmienna lokalna) double number

Tym razem użyta została skrócona nazwa typu (**double**).

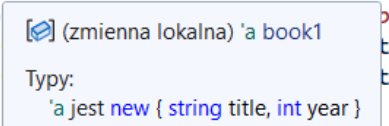
Typ anonimowy

Niejawny typ danych jest bardzo przydatny do tworzenia krótkotrwałych obiektów, dla których nie chcemy tworzyć klasy. Typ taki jest z zasady używany głównie w sytuacji, gdy jest tworzony i używany w ramach jednej metody. Obiekty tego typu tworzy się poprzez operator `new` (bez nazwy klasy), po którym następują nawiasy klamrowe, w których wewnątrz definiuje się pola takiego obiektu. Definicje pól rozdzielane są przecinkiem, a jedno pole to identyfikator pola, znak równości i wyrażenie definiujące jego zawartość. Dostęp do pól jest realizowany za pomocą operatora kropki (jak dla innych obiektów) np.:

```
var book1 = new { title = "The War of the Worlds", year = 1898 };
var book2 = new { title = "The Hobbit or There and Back Again", year = 1937 };
System.Console.WriteLine($"{book1.title} from year {book1.year}");
System.Console.WriteLine($"{book2.title} from year {book2.year}");
```

Kompilator dla obiektów takiego typu wytwarza specjalne typy i dlatego nie można zmiennych typów anonimowych zadeklarować inaczej jak z pomocą `var`. Jeśli nazwy pól i ich typ dla kilku takich zmiennych są takie same, to kompilator wytworzy jeden wspólny typ. Jednak nie ma dostępu do tej informacji w IDE, gdyż nazwa takiego typu jest zawsze prezentowana jako `'a`. Jednak w dolnej części okienka pojawia się informacja o nazwach pól i ich typach.

```
var book1 = new { title = "The War of the Worlds", year = 1898 };
var book2 = new { title = "The Hobbit or There and Back Again", year = 1937 };
System.Console.WriteLine($"{book1.title} from year {book1.year}");
System.Console.WriteLine($"{book2.title} from year {book2.year}");
```



Typ anonimowy w wielu sytuacjach zastąpiony może być przez krotkę.

Krotki

Krotki w języku C# zapisuje się jako zestaw danych w nawiasach, rozdzielone przecinkami. W odróżnieniu od typu anonimowego dane w krotce najczęściej nie posiadają nazwy (**krotki nienazwane**). Przykład:

```
("Hothouse", "Brian W. Aldiss", 120.0)
```

Aby dostać się składowych takiej krotki można użyć predefiniowanych właściwości o nazwach `Item1`, `Item2`, itd. Więc zakładając, że krotka jest przechowana w zmiennej np.:

```
var bookInfo7 = ("Hothouse", "Brian W. Aldiss", 120.0);
```

Można odczytać kolejne pola poprzez:

```
System.Console.WriteLine($"Book {bookInfo7.Item1} by {bookInfo7.Item2} for {bookInfo7.Item3} zł.");
```

Nie jest to jednak najlepszy sposób korzystania z krotki. Jednym z lepszych sposobów to nadanie nazw polom krotki, co można zadeklarować w zapisie zbliżonym do krotki. Jednak zamiast wartości podane są deklaracje pól, czyli nazwa typu i nazwa pola. W dodatku te pola krotki, które nie będą potrzebne można zastąpić „jokerem”, czyli znakiem pokreślenia `'_'`.

Przykład:

```
(string Name, string _, double Price) bookInfo5 = ("Hothouse", "Brian W. Aldiss", 120.0);
System.Console.WriteLine($"Book {bookInfo5.Name} for {bookInfo5.Price} zł.");
```

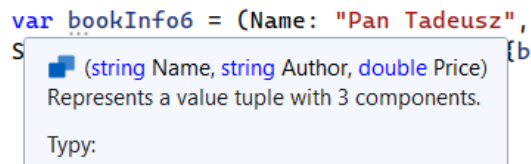
Inny ciekawy sposób to deklaracja zmiennych, do których zostaną przyporządkowane kolejne pola. Deklaracje zmiennych można wykonać na wiele sposobów, efekt jest zawsze taki sam:

```
var (name4, author4, price4) = ("Hothouse", "Brian W. Aldiss",
120.0);
System.Console.WriteLine($"Book {name4} by {author4} for
{price4} zł.");
(string Name, string Author, double Price) bookInfo5 =
("Hothouse", "Brian W. Aldiss", 120.0);
System.Console.WriteLine($"Book {bookInfo5.Name} by
{bookInfo5.Author} for {bookInfo5.Price} zł.");
```

Krotka nazwana jest jeszcze bardziej zbliżona do typu anonimowego. W tym przypadku przed każdą wartością krotki znajduje się nazwa pola oraz dwukropek, np.:

```
var bookInfo6 = (Name: "Hothouse", Author: "Brian W. Aldiss",
Price: 120.0);
System.Console.WriteLine($"Book {bookInfo6.Name} by
{bookInfo6.Author} for {bookInfo6.Price} zł.");
```

Jednak sprawdzając w środowisku IDE typ otrzymujemy:



```
var bookInfo6 = (Name: "Pan Tadeusz",
S (string Name, string Author, double Price) {b
Represents a value tuple with 3 components.
Typy:
```

Czyli informację, że jest to krotka (ang. tuple) a nie jakiś typ `a`.

Moduły wbudowane

Język C# ma mnóstwo modułów w ramach biblioteki `System`. Są to moduły dostępne w każdym typie projektu, więc można nazwać je modułami wbudowanymi. Część z nich służy do operowania na kolekcjach obiektów (tablice, list wiązanych, drzew, tablic mieszanych itd.).

Celem tej listy jest zapoznanie się samemu z wybraną kolekcją i jej metodami.

Instrukcja switch/case

Instrukcja **switch/case** obecna jest w wielu językach programowania. W większości z nich w części **switch** należy podać wyrażenie typu prostego (ewentualnie typu **string**), którego wartość będzie decydowała, która z kolejnych instrukcji **case** będzie wykonana. Sama instrukcja **case** składa się ze stałej wartości zgodnej z typem części **switch** oraz dwukropka po którym następuje ciąg instrukcji do wykonania. Ciąg instrukcji powinien być jedną z instrukcji: **break** lub **goto**. Wersja **goto** musi być z kolejnym słowem **default** albo **case** (ze stałą). Dopuszczalna jest również jedna sekcja **default**, która wykonuje się, gdy żaden przypadek z **case** nie pasuje.

```
static void SwitchInstruction(string line)
{
    switch (line)
    {
        case "Good":
        case "OK": Console.WriteLine("All right");
```

```

        break;
    case "Super": Console.Write("Extra - ");
        goto case "OK";
    case "Can be":
        Console.WriteLine("It is fine");
        break;
    default:
        Console.WriteLine("I don't understand...");
        break;
    }
}
static void SwitchInstructionTest()
{
    SwitchInstruction ("Good");
    SwitchInstruction ("Can be");
    SwitchInstruction ("Super");
    SwitchInstruction ("So-so");
}

```

W języku C# jako wyrażenie w **switch** może być wyrażenie typu generycznego, co powoduje, że w sekcjach **case** muszą być deklaracje zmiennych konkretnych typów. Można wówczas dodać klauzulę **when**, w której umieszcza się wyrażenie logiczne (z użyciem zmiennej z danej części **case**), która musi być prawdą, aby instrukcje zostały wykonane.

Przykład:

```

private static void ShowShapeInfo(Shape sh){
    switch (sh)
    {
        // Note that this code never evaluates to true.
        case Shape shape when shape == null:
            Console.WriteLine($"An uninitialized shape (shape == null)");
            break;
        case null:
            Console.WriteLine($"An uninitialized shape");
            break;
        case Shape shape when sh.Area == 0:
            Console.WriteLine($"The shape: {sh.GetType().Name} with no dimensions");
            break;
        case Square sq when sh.Area > 0:
            Console.WriteLine("Information about square:");
            Console.WriteLine($"    Length of a side: {sq.Side}");
            Console.WriteLine($"    Area: {sq.Area}");
            break;
        case Rectangle r when r.Length == r.Width && r.Area > 0:
            Console.WriteLine("Information about square rectangle:");
            Console.WriteLine($"    Length of a side: {r.Length}");
            Console.WriteLine($"    Area: {r.Area}");
            break;
    }
}

```

List zadań

1. Stworzyć krotkę (z użyciem nawiasów okrągłych, jak na wykładzie, nie używać krotek z zapisem klasy generycznej!) składającą się z danych: imię, nazwisko, wiek, płać. Wywołaj funkcję z tą krotką **jako parametrem**. Na 3 sposoby pokaż możliwość skorzystania z tej krotki, aby wypisać podane informacje na konsoli.
2. Stwórz zmienną o nazwie `class`, zapisz do niej jakąś wartość i wypisz na ekranie.
3. Zaprezentuj użycie wybranych 5 różnych metod z modułu `System.Arrays`.
4. Czy zadanie 1 można wykonać z użyciem typu anonimowego w tak samo prosty sposób?
5. Napisać metodę `DrawCard` „rysującą” wizytówkę. Metoda otrzymuje w parametrach pierwszą linię wizytówki, drugą linię, znak obramowania, szerokość obramowania, minimalną szerokość całej wizytówki. **Oprócz pierwszego** parametru pozostałe są **opcjonalne** (wybrać własne domyślne wartości). Zademonstrować **różne** wywołania metody w tym z użyciem **parametrów nazwanych**. Wizytówka to prostokątna ramka z wycentrowanymi napisami np. dla zestawu danych `DrawCard („Ryszard”, „Rys”, „X”, 2, 20)` „narysuje”:

```
XXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXX
XX      Ryszard      XX
XX      Rys         XX
XXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXX
```

6. Stworzyć metodę `CountMyTypes`, w której parametrach można podać **dowolny ciąg elementów rozdzielonych przecinkami**, a metoda ta zliczy ile wśród parametrów było parzystych liczb całkowitych (`int`), liczb rzeczywistych (`double`) dodatnich, napisów co najmniej 5-znakowych i elementów innych typów niż `w/w`, zwracając te **cztery wartości** w wybrany sposób. Użyć instrukcji `switch/case`.

Data I: Spotkanie 7 (100 punktów)

Data II: Spotkanie 8 (80 punktów)

Data III: Spotkanie 9 (50 punktów)