Wyższa Szkoła Bankowa
Wydział Finansów i Zarządzania
Informatyka Inżynierska
Studia I stopnia, semestr 2
Gdańsk

Programowanie obiektowe
Laboratorium 3 & 4

Autor:
Jakub Głuszek

24 kwietnia 2020

Spis treści

1 Lab 3 **3**
1.1 Inicjalizacja struktury 3
1.2 Struktury DateTime, DateTimeOffset oraz TimeSpan 3
1.3 Porównanie struktur oraz klas 4

2 Lab 4 **5**
2.1 Indeksatory 5
2.2 Składowe statyczne 6
2.3 Klasy statyczne 7

1 Lab 3

1.1 Składowe statyczne

Analogicznie do poprzednich laboratoriów utwórz klasę `Employee`. Niech zawiera ona dwie właściwości np. `Name` oraz `Id` (w tym momencie nie będą nam one potrzebne). Dodatkowo do ww. klasy dodaj jedno pole (albo właściwość) o nazwie `EmployeeCounter` i oznacz je jako statyczne[1] - w tym celu wykorzystaj słowo kluczowe `static`.

```
class Employee()
{
  public /*?*/ Name {get;set;}
  public /*?*/ Id {get;set;}
  public static int EmployeeCounter {get; set} //as a field/property, will
    be set to zero by default
}
```

Dodaj do tej klasy konstruktor w którym będziesz inkrementował wartość licznika `EmployeeCounter` przy każdym tworzeniu nowego obiektu typu `Employee`.

```
public Employee()
{
  //...
}
```

W metodzie `Main` odczytaj właściwość `EmployeeCounter` (zauważ, że możesz to zrobić bez tworzenia nowej instancji typu `Employee`). Następnie stwórz dwa obiekty pracowników i odczytaj ponownie wartość `EmployeeCounter`.

```
Console.WriteLine(Employee.employeeCounter);
Employee employee1 = new Employee();
Employee employee2 = new Employee();
Console.WriteLine(Employee.employeeCounter);
```

Tak samo jak pole czy właściwość, statyczna może być metoda. Dodaj do klasy `Employee` statyczną metodę `PrintEmployeeCounter()`, która będzie wyświetlać liczbę pracowników.

```
public static void PrintEmployeeCounter()
{
  Console.WriteLine($"There're {employeeCounter} employees");
}
```

Spróbuj teraz zamiast statycznej właściwości `EmployeeCounter` w metodzie `PrintEmployeeCounter()` użyć dowolnej innej niestatycznej właściwości np. `Id`
Czy i dlaczego kompilator na to (nie)pozwoli? (odpowiedź umieść w komentarzu)

Dodaj do klasy `Employee` statyczny konstruktor[2] informujący o inicjalizacji typu.

```
static Employee() //static constructor can not be public or private, since
    it cannot be called manually
{
  Console.WriteLine("Type has been initialized");
}

//or shorter
```

[1] Składowa statyczną należy do samego typu, a nie konkretnego obiektu. Można się do niej odwoływać bez tworzenia obiektu danego typu. Inicjalizatory pól są wykonywane bezpośrednio przed uruchomieniem konstruktora, albo jeżeli taki konstruktor nie został zadeklarowany to w dowolnym momencie przed użyciem typu (od uruchomienia programu do użycia tego typu). Oznaczenie klasy jako statycznej (przy użyciu słowa kluczowego `static` sprawia, że wszystkie składowe takiej klasy muszą być statyczne). Klasy statycznę są „zapieczętowane" oznaczone jako `sealed` oznacza to, że niemożliwe jest dziedziczenie po klasie statycznej. Nie mogą one również dziedziczyć po żadnej klasie z wyjątkiem niejawnego dziedziczenia po klasie `Object`.

[2] Statyczny konstruktor jest wykonywany tylko raz dla typu, a nie dla każdego egzemplarza. W typie może być zdefiniowany tylko jeden taki konstruktor i nie może on przyjmować parametrów.

```
static Employee() => Console.WriteLine("Type has been initialized");
```

CLR wykona kod takiego konstruktora przed utworzeniem egzemplarza typu, albo przy dostępie do statycznej składowej typu.

UWAGA! Ważne, aby taki konstruktor nie zgłosił wyjątku, bo wtedy typ staje się bezużyteczny do końca działania aplikacji.

1.2 Klasy statyczne

Jako statyczną można również oznaczyć klasę. Dobrymi przykładami klas statycznych są `System.Console` czy `System.Math`. W obu przypadkach tworzenie instancji tych klas nie miałoby sensu zakładamy, że w programie będzie istniała tylko jedna konsola, a funkcje matematyczne nie są związane z żadnym konkretnym obiektem.

Stwórz klasę która będzie dokonywała konwersji stopni Fahrenheita na stopnie Celsjusza. Łatwo zauważyć, że w tym przypadku tworzenie instancji takiej klasy nie miałoby sensu.

```
public static class TemperatureConverter
{
  public static double CelsiusToFahrenheit(double temperatureCelsius)
  {
    return (temperatureCelsius * 9 / 5) + 32;
  }

  public static double FahrenheitToCelsius(double temperatureFahrenheit)
  {
    return ...
  }
}
```

Sprawdź poprawność dokonywanej konwersji (w obie strony) w metodzie `Main()`. Np. 20 stopni w skali Fahrenheita odpowiada -6.67 stopniom w skali Celsjusza.

1.3 Struktury DateTime, DateTimeOffset oraz TimeSpan

Do określania dat i godzin służą struktury takie jak: `DateTime`[3], `DateTimeOffset`[4] i `Timespan`[5].

Stwórz obiekt typu `TimeSpan` - sprawdź jakie ma przeciążenia konstruktora. Wypisz na ekranie konsoli wartości przechowywane we właściwościach: `TotalMinutes`, `Minues` - przejrzyj jakie inne składowe udostępnia ten typ. Zobacz jak działają metody `FromHours()` oraz `Parse()`.

```
TimeSpan timeSpan = new TimeSpan(1, 12, 30, 30);
Console.WriteLine(timeSpan.TotalMinutes);
Console.WriteLine(timeSpan.Minutes);
Console.WriteLine(TimeSpan.FromHours(100));
Console.WriteLine(TimeSpan.Parse("12.12:21:21"));
```

Analogicznie utwórz obiekty typów `DateTime` oraz `DateTimeOffset`. Sprawdź jakie inne składowe udostępnia ten typ. Obiekty tego typu można tworzyć za pomocą konstruktora (przekazując odpowiednie parametry), można też skorzystać z statycznej właściwości `Now` albo `UtcNow`, które zwrócą aktualny czas z albo bez uwzględnienia przesunięcia czasowego. Drugi sposób przedstawiono poniżej:

```
DateTimeOffset dateTimeOffset = DateTimeOffset.UtcNow;
DateTime dateTime = DateTime.Now;
```

Sprawdź (zwracane) wartości kilku ich składowych np. `DayOfWeek`, `DayOfYear`, `ToShortTimeString()` czy `IsDaylightSavingTime()`

[3] Do operacji związanych z czasem w `.NET` można wykorzystać strukturę `DateTime`. Reprezentuje ona chwilę w czasie, zwykle wyrażoną jako datę i godzinę.

[4] Posiada analogiczne funkcjonalności jak `DateTime`, jednak dodatkowo przechowuje informację o przesunięciu czasu UTC

[5] Reprezentuje pewien przedział czasu. Dokładność obiektów tego typu wynosi 100 ns

```
Console.WriteLine(DateTime.Now.DayOfWeek);
Console.WriteLine(DateTime.Now.ToShortTimeString());

Console.WriteLine(DateTime.Now.IsDaylightSavingTime());
Console.WriteLine(new DateTime(2020,01,01,12,0,0).IsDaylightSavingTime());
```

Na obiektach omawianych typów można wykonywać obliczenia. Sprawdź działanie przeciążonego operatora + (działa on tak samo jak metoda `Add()`).

```
Console.WriteLine((DateTime.Now + TimeSpan.FromDays(2)).DayOfWeek);
Console.WriteLine(TimeSpan.FromHours(100) - timeSpan);
```

Ponieważ `DateTime` jest strukturą operator == wykorzystuje on porównanie wartościowe. Stwórz dwa obiekty przechowujące informację o „pewnej" chwili w czasie. Przekaż w obu przypadkach do konstruktora takie same wartości.

```
DateTime dateTime1 = new DateTime(2020,1,1,12,0,0);
DateTime dateTime2 = new DateTime(2020,1,1,12,0,0);
```

Wykonaj porównanie obu obiektów:

```
...
Console.WriteLine(dateTime1==dateTime2); //To compare amount of 100 ns
    ticks have been taken
...
DateTime local = DateTime.Now;
DateTime utc = DateTime.Now.ToUniversalTime();
Console.WriteLine(local == utc); //true/false
```

2 Lab 4

2.1 Inicjalizacja struktury

Stwórz dowolną strukturę[6] posiadającą przynajmniej dwa pola. Struktury definiujemy przy pomocy słowa kluczowego `struct`.

```
public struct PointStruct
{
  public int X {get;set;}
  public int Y {get;set;}
}
```

W metodzie `Main` utwórz obiekt struktury (tworzymy go w analogiczny sposób do tworzenia obiektów klas) i wyświetl na ekranie konsoli domyślne wartości jego właściwości. Następnie zmień ich wartości i ponownie wyświetl na ekranie.

Dodaj do wcześniej zdefiniowanej struktury konstruktor[7], w którym polom struktury zostaną przypisane wartości przekazane jako argumenty tego konstruktora.

```
public struct PointStruct
{
  public int X {get;set;}
  public int Y {get;set;}
  public PointStruct(int x, int y) {this.X = x; this.Y=y;}
}
```

Analogicznie w metodzie `Main()` utwórz kolejny obiekt, wykorzystując stworzony przed chwilą konstruktor.

[6] Struktury w przeciwieństwie do klas są typami wartościowymi (nie referencyjnymi). Nie mogą one dziedziczyć po innych strukturach (choć nie jawnie pochodzą od klasy `System.ValueType`). Warto jednak pamiętać, że struktury mogą implementować interfejsy.

[7] Struktura może posiadać konstruktor, w takim przypadku **każde** pole musi mieć jawnie przypisaną wartość.

2.2 Porównanie struktur oraz klas

W podobny do struktury `PointStruct` sposób, stwórz **klasę** `PointClass`.

```
class PointClass{public int x,y;}
```

W metodzie `Main()` stwórz dwie instancje typu `PointStruct` (zdeklarowanej wcześniej struktury) oraz dwie instancje typu `PointClass` (zdeklarowanej wcześniej klasy).

```
PointStruct pointStructA = new PointStruct();
PointStruct pointStructB = new PointStruct();
PointClass pointClassA = new PointClass();
PointClass pointClassB = new PointClass();
```

W każdym z obiektów przypisz właściwościom/polom takie same wartości.

```
...
pointStructA.X = 1;
pointStructA.Y = 1;
...
pointClassA.X = 1;
pointClassA.Y = 1;
```

Porównaj [8] obiekt `pointStructA` z `pointStructB` oraz obiekt `pointClassA` z `pointClassB`, wykorzystaj w tym celu metodę `Equals()`[9]

```
...
Console.WriteLine(pointStructA.Equals(pointStructB));
Console.WriteLine(pointClassA.Equals(pointClassB));
...
```

Sprawdź jakie otrzymałeś wyniki.
Dlaczego porównując dwie klasy i struktury posiadające takie same wartości pól/właściwości otrzymamy inne wyniki? (odpowiedź umieść w komentarzu)

Dodaj nowy obiekt `PointClassC` i przypisz do niego obiekt `PointClassA`. Wykonaj analogiczne porównanie obu klas.

```
...
PointClass pointClassC = pointA;
Console.WriteLine(pointClassA.Equals(pointClassC));
...
```

Ponownie sprawdź wyniki.
Dlaczego teraz porównując dwa obiekty klasy `PointClass` otrzymaliśmy inny wynik? (odpowiedź umieść w komentarzu)

Stwórz dwie metody, które będą zmieniać jedną ze składowych obiektów typ `PointStruct` oraz `PointClass`.

```
...
public static void PointChanger(PointClass point)
{
  //...
}
```

[8] Można rozróżnić dwa rodzaje równości:

- wartościowa - dwie wartości są równe w jakimś sensie
- referencyjna - dwie referencje odnoszą się dokładnie do tego samego obiektu

Z typami wartościowymi domyślnie wykorzystywane jest porównanie wartościowe, natomiast z typami referencyjnymi porównanie referencyjne.

[9] Wirtualna metoda Object.Equals jest rozpoznawana podczas działania programu (w przeciwieństwie do operatorów == oraz != , które podlegają rozpoznawaniu statycznemu). Jeżeli porównywane będą typy wartościowe (np. typ `Object` będzie przechowywał wartość typu `Int32`) zostanie wykorzystane porównanie wartościowe. Dla typów referencyjnych porównanie będzie referencyjne.

```
...
public static void PointChanger(PointStruct point)
{
  //...
}
```

Przekaż teraz wcześniej utworzone w metodzie `Main` obiekty do tych metod i sprawdź jakie wartości mają składowe tych obiektów po „wyjściu" z tych metod t.j. `PointChanger`.
Jakie zaobserwowałeś różnice? Dlaczego nastąpiła zmiana jednej z wartości? (odpowiedź umieść w komentarzu)

Prawie zawsze lepiej jest tworzyć klasy niż struktury. Struktury mogą okazać się pomocne jeśli:

- **przy przypisywaniu pewnej wartości skopiowanie jest bardziej naturalne, aniżeli przypisanie referencji,**
- **instancje są małe i zazwyczaj mają krótki czas życia,**
- **tych instancji jest bardzo dużo.**

2.3 Indeksatory

Do deklaracji indeksatora[10] używamy słowa kluczego `this`. Aby zadeklarować indeksator dla klasy lub struktury, użyj słowa kluczowego `this`[11] , jak pokazano w poniższym przykładzie:

```
class Sentence
{
  string[] words = "Talk is cheap. Show me the code.".Split();

  public string this [int wordNum] //indexer
  {
    get {return words[wordNum]; }
    set { words [wordNum] = value; }
  }
}
```

Utwórz w `Program` instancję klasy `Sentance` i spróbuj odwołać się do jej składowej tak jak do tablicy poprzez indeks. Następnie również przy użyciu indeksu zmień jedną wartość w tablicy `words` i ponownie ją wypisz na ekranie konsoli.

```
Sentence s = new Sentence();
Console.WriteLine(s[2]); //cheap.
// ...
// ...
```

Przykładem klas, które posiadają zaimplementowane indeksatory są np. kolekcje, ale można z nich również korzystać w innych typach np. `System.String`.

Stwórz dowolny łańcuch znaków i spróbuj odwołać się do niego poprzez indeksator.

```
string str = "We're done for today!";
for (int counter = 0; counter < str.Length; counter++)
{
  Console.WriteLine(nameString[counter]);
}
```

[10] Indeksatory są podobne do właściwości, ale dostęp do nich odbywa się poprzez argument indeksowy. Możemy z nich korzystać jeżeli nasza klasa potrzebuje listy albo tablicy własnych instancji albo nasza klasa reprezentuje pewną listę albo tablice.

[11] Słowo kluczowe `this` posiada kilka zastosowań. Może oznaczać odwoływanie się do **tej** konkretnej instancji klasy/struktury. Słowa kluczowego `this` używamy również, aby przekazać instancję klasy jako parametr metody. `This` jest również używane przy tworzeniu indeksatorów.

Bibliografia

[1] Ben Albahari Joseph Albahari. *C# 7.0 w pigułce*. Helion, Gliwice 2018.

[2] Microsoft. *Static Classes and Static Class Members*. [Online; 11-04-2020]. URL: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/static-classes-and-static-class-members.

[3] *Choosing Between Class and Struct*. [Online; 11-04-2020]. URL: https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/choosing-between-class-and-struct.