FT Training & Software

C# 3.0

Tomasz Toboła
tomasz.tobola@ftts.pl
+48 601-263-904

C# 3.0

C# Language Specification 3.0
http://msdn.microsoft.com/en-us/vcsharp/aa336809.aspx

- Słowo kluczowe var
- Konstruktor kolekcji
- Konstruktor z listą właściwości
- Typy anonimowe
- Rozszerzenia klas
- Metody częściowe
- Wyrażenia Lambda
- LINQ (Language INtegrated Query)

Słowo kluczowe var

C# 3.0

Słowo kluczowe var służy do zadeklarowania zmiennej niewiadomego typu. Typ będzie określony na podstawie analizy kodu.

```
var p = new Pracownik();
    p.Nazwisko = "Kowalski";
```


Konstruktor kolekcji

C# 3.0

Upraszcza tworzenie kolekcji z wcześniej znanymi wartościami

```
List<string> listaDni = new List<string>() {
   "Poniedziałek",
   "Wtorek"
};

Dictionary<int, string> slownikProduktow =
   new Dictionary<int, string>() {
       { 1, "Proszek" },
       { 2, "Pasta" },
       { 3, "Szampon" }
   };
```

Konstruktor z listą właściwości

C# 3.0

Pozwala przypisać wartości podczas tworzenia obiektu. Przydatny dla klas bez konstruktora z listą parametrów, których wartości są przepisywane do pól.

```
class Pracownik
{
    public string Imie { get; set; }
    public string Nazwisko { get; set; }
}
Pracownik mojPracownik = new Pracownik()
{
    Imie = "Adam",
    Nazwisko = "Kowalski"
};
```


Typy anonimowe

C# 3.0

Pozwala zdefiniować typ referencyjny w ciele metody;

```
static void Main()
{
   var pracownik = new { Imie = "Adam", Nazwisko = "Kowalski" };
   Console.WriteLine(pracownik.Imie + " " + pracownik.Nazwisko);
}
```

Rozszerzenia klas

C# 3.0

Mechanizm pozwala na dodawanie własnych metod do istniejących klas. Użycie rozszerzenia poprzedza zaimportowanie odpowiedniej przestrzeni nazw.

```
static class Walidator
{
  public static bool CzyJestPoprawny(this string wartosc, string wzorzecRegex)
  {
      return Regex.IsMatch(wartosc, wzorzecRegex);
  }
}
string Nip = "111-222-33-44";
if (!Nip.CzyJestPoprawny(@"^\d{3}-\d{3}-\d{2}-\d{2}$"))
{
   MessageBox.Show("Popraw numer NIP");
   return;
}
```


Metody częściowe

C# 3.0

Mechanizm podobny do C++ (deklaracja oraz definicja metody)

```
//plik Pracownik.cs
partial class Pracownik
{
    partial void Zapisz();
    public string Imie { get; set; }
    public string Nazwisko { get; set; }
}

//plik PracownikDB.cs
partial class Pracownik
{
    partial void Zapisz()
    {
        ...
    }
}
```

Lambda expressions

C# 3.0

```
// C# 1.0 delegat zwykły
static class Program
{
   delegate int delegatDodawanie(int x, int y);
   static void Main()
   {
        delegatDodawanie funkcjaDodajaca = Dodaj;
        Console.WriteLine(funkcjaDodajaca(4, 2));
   }
   static int Dodaj(int x, int y)
   {
        return x + y;
   }
}
```


Lambda expressions

C# 3.0

```
// C# 2.0 delegat anonimowy
static class Program
{
   delegate int delegatDodawanie(int x, int y);
   static void Main()
   {
        delegatDodawanie funkcjaDodajaca =
                                delegate (int x, int y) { return x + y; } ;

        Console.WriteLine(funkcjaDodajaca(4, 2));
   }
}
```

Lambda expressions

C# 3.0

```
// C# 3.0 lambda expressions
static class Program
{
   delegate int delegatDodawanie(int x, int y);
   static void Main()
   {
        delegatDodawanie funkcjaDodajaca = (x, y) => x + y;

        Console.WriteLine(funkcjaDodajaca(4, 2));
   }
}
```


Lambda expressions

C# 3.0

Praca z delegatami, uproszczenia:

```
static class Program
{
   static void Main()
   {
        Func<int, int, int> funkcjaDodajaca = (x, y) => x + y;
        Console.WriteLine(funkcjaDodajaca(4, 2));
   }
}

Func<T1,...,T4, TResult>  // funkcja zwraca TResult
Action<T1,...,T4>         // funkcja VOID
Predicate<T>              // funkcja zwraca bool
```

LINQ

C# 3.0

Zintegrowany język zapytań

http://msdn.microsoft.com/en-us/netframework/aa904594.aspx

```
static void Main()
{
      int[] liczby = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
      var liczbyParzyste = liczby.Where(n => n % 2 == 0);
      foreach (var liczba in liczbyParzyste)
      {
          Console.WriteLine(liczba);
      }
      // lub
      liczbyParzyste.ToList().ForEach(l => Console.WriteLine(l));

      Console.WriteLine(liczbyParzyste.Max());
      Console.WriteLine(liczbyParzyste.Min());
      Console.WriteLine(liczbyParzyste.Average());
}
```


LINQ – operatory zapytań

C# 3.0

Select	OfType	FirstOrDefault
SelectMany	Concat	Last
Where	OrderBy	LastOrDefault
Sum	OrderByDescending	Single
Min	ThenBy	ElementAt
Max	ThenByDescending	Any
Average	Reverse	All
Aggregate	GroupBy	Contains
Join	Distinct	Count
GroupJoin	Union	
Take	Intersect	
TakeWhile	Except	
Skip	EqualAll	
SkipWhile	First	