Curs09

Metode extinse (Extension methods)

O metoda care `extinde` metodele publice deja existente pentru o anumită clasă căreia nu i se poate modifica codul sursă. (Este într-o librărie la care nu avem acces).

Se declară într-o clasă statică, de tip static. Sintaxă generală:

```
public static Extensions {
  public static[tip_returnat] MethodName(this NumeClasaExtinsa numeRef,
[lista_param]) {
    //...
  }
}
```

Sintaxă de apelare: `numeRef.MethodName([lista_param]);`

Exemplu

1. Să se extindă clasa `string` (deja existentă) cu o metodă care verifică câte cifre apar în stringul respectiv.

```
public static class Extensions {
  public static int GetDigitsCount(this string textValue) {
    if (string.IsNullOrEmpty(textValue))
      return 0;
    int count = 0;
    foreach(var c in textValue)
      if (char.IsDigit(c))
        count++;
    return count;
  }
}

class Program {
  static void Main(string[] args) {
    string s = "adasfasr3443f afa 45 ";
    var nr = s.GetDigitsCount();

    Console.WriteLine("Numarul de cifre: ", nr);
    Console.ReadKey();
  }
}
```

2. Se consideră clasa deja existenta `List<T>`. Să se adauge o metodă extinsă care verifică dacă un obiect de tip `List<T>` este `null` sau este lista goală (`IsNullOrEmpty`).

```
public static bool IsNullOrEmpty<T> (this List<T> list) {
  if (list == null)
    return true;
  if (list.Count == 0)
    return true;
  return false;
}
```

3. Să se adauge clasei `List<T>` o metodă care verifică de câte ori apare o valoare dată în listă `GetOccurrences`.

```
public static int GetOccurrences<T> (this List<T> list, T value) {
  int result = 0;
  foreach(var v in list)
    if (v.Equals(value))
      result++;
  return result;
}

List<string> l = new List<string>() { "aasda", "as", "asda", "as", "asdasdas",
"as" };
Console.WriteLine(l.GetOccurrences<string>("as"));
```

Introducere în LINQ to Objects

`LINQ (Language Integrated Query)` permite interogarea colecţiilor de obiecte cu operatori asemănători cu cei din `SQL` (`SELECT`, `WHERE`, `ORDERBY`,...).

Sunt implementaţi ca metode extinse in libraria `LINQ`:

```
List<string> list = new List<string>() { "Aasda", "as", "asda", "as", "Asdasdas",
"as" };
var values = list.Where(x => char.IsUpper(x[0])).ToList();
```

Observaţie: pentru a putea utiliza `LINQ` pentru colecţii de obiecte, acestea trebuie să fie `enumerabile` (adică să implementeze interfaţa `IEnumerable`, `IEnumerable<T>`). Operatorul `WHERE` (filtrarea obiectelor colecţiei cu un anumit predicat).

```
class Program {
  class Car {
    public string Model { get; set; }
    public int Year { get; set; }
    public string Color { get; set; }
  }
  static void Main(string[] args) {
    // Operatorul WHERE
    // List<int> values = new List<int>() { 3, 2, 5, 6, 7, 8, 35, 7 };
    // var values1 = values.Where(x => x % 2 == 0).ToList();
    List<Car> cars = new List<Car>() {
      new Car() { Model = "bmw", Year = 2000, Color = "red" },
      new Car() { Model = "toyota", Year = 2018, Color = "black" },
      new Car() { Model = "mercedes", Year = 2016, Color = "red" },
      new Car() { Model = "dacia", Year = 2013, Color = "black" },
      new Car() { Model = "bmw", Year = 2012, Color = "blue" },
```

```
      new Car() { Model = "bmw", Year = 2010, Color = "green" }
    };

    var c1 = cars.Where(x => x.Color == "red" && x.Year> 2000).ToList();
    Console.ReadKey();
  }
}
```

Implementare:

```
public static IEnumerable<TSource> Where<TSource> (this IEnumerable<TSource>
source, Func<TSource, bool> predicate) {
    // ...
}
```

Implementare proprie a unui operator `Where`:

```
public static class MyExtensionsOperators {
  public static List<T> MyWhere<T> (this List<T> source, Func<T, bool> predicate)
{
    var result = new List<T>();
    foreach(var elem in source)
      if (predicate(elem))
        result.Add(elem);
    return result;
  }
}
```

Operatorul `Select` (proiecţie) - Selectăm doar anumite proprietăţi ale obiectelor.

Implementare

```
public static IEnumerable<TResult> Select<TSource, TResult> (this
IEnumerable<TSource> source, Func<TSource, TResult> selector);

public static class MyExtensionsOperators {
  public static List<T> MyWhere<T> (this List<T> source, Func<T, bool> predicate)
{
    var result = new List<T>();
    foreach(var elem in source)
      if (predicate(elem))
        result.Add(elem);
    return result;
  }

  public static List<TResult> MySelect<T, TResult> (this List<T> source, Func<T,
TResult> selector) {
    var result = new List<TResult>();
    foreach(var elem in source)
      result.Add(selector(elem));
    return result;
  }
}
```