Extensions du langage C#

Le langage C# est un langage relativement riche, toutefois on peut encore l'enrichir selon les besoins spécifiques d'un projet ou d'une entreprise...

D'ailleurs, LINQ utilise ce mécanisme pour étendre les possibilités des `IEnumerable`, par exemple si on regarde la signature de `Where` :

```
public static System.Linq.IQueryable<TSource> Where<TSource> (/*C'est quoi ça ?*/ this /*?????*/
System.Linq.IQueryable<TSource> source, System.Linq.Expressions.Expression<Func<TSource,bool>>
predicate);
```

La *magie* des extensions consiste à définir une fonction `statique et publique` et définir comme premier paramètre un élément du type qu'on veut étendre et préfixé par le mot clé `this`. Et tout ça dans une classe elle aussi `public et statique` :

```
public static class PeopleExtensions
{
    public static string Greetings(this string name)
    {
        return $"Hello {name}";
    }
}
```

Ce code étend la classe string de C# en ajoutant une méthode `Greetings()` :

```
Console.WriteLine("Bob".Greetings());
```

Ce code affichera

```
Hello Bob
```

Pour que vos méthodes d'extension soient utilisables, il faut que la classe qui les définit se trouve

- Soit dans le même namespace que votre code
- Soit dans un namespace importé avec `using`

Un peu mieux

En ajoutant une méthode d'extension qui ne retourne rien (`void`), on peut faire encore mieux.

Méthode supplémentaire :

```
public static void ToConsole(this string subject)
{
    Console.WriteLine(subject);
}
```

Utilisation :

```
"Bob".Greetings().ToConsole();
```

> *Cet exemple montre clairement que le chaînage fonctionnel part des données (ici "Bob"). Ainsi c'est un point de vue qui peut être déstabilisant, car on avait l'habitude jusque-là de choisir d'abord l'action (avec Console.Write)...*

Chaînage avec valeur de retour

Quand une méthode d'extension retourne un élément, cela permet de faire du chaînage. Avec la méthode suivante :

```
    /// <summary>
    /// Passe en minuscule les éléments fournis, éventuellement de manière aléatoire
    /// </summary>
    /// <param name="subject">Les données en input</param>
    /// <param name="random">Si vrai, aura 50% de faire la transformation</param>
    /// <returns></returns>
    public static IEnumerable<string> ToLower(this IEnumerable<string> subject,bool random=false)
    {
        return subject.Select(text =>
            random?
                randomGenerator.Next(2)==1?
                    text.ToLower():text
                :text.ToLower());
    }
```

> *On constate dans cet exemple qu'une méthode d'extension peut avoir des paramètres standards à la suite du premier (ici bool random=false)...*

On peut écrire le code suivant :

```
var data = new[] {"BoB","Max","jOelLe","NadiA" };
data
    .Where(name => name.StartsWith("j"))
    .ToLower(true)
    .ToList()
    .ForEach(Console.WriteLine);
```

▶ Quel sera le résultat ?

DSL : *D*omain *S*pecific *L*anguage

Un exemple concret de l'utilisation des extensions est de pouvoir créer un `langage spécifique au domaine`. On entend par là un pseudo-langage qui est plus proche du métier.

Exemple 1 : FluentAssertions

Ceci a été utilisé pour la librairie d'assertion pour les tests `FluentAssertions`, qui permet d'écrire les assertions ainsi :

```
[Fact]
public void TestIsMatch()
{
    //Arrange
    var cmd = "--help";

    //Act
    bool result = cmd.IsMatch(Program.OptHelp);

    //Assert
    result.Should().BeTrue("la ligne de commande contient le paramètre --help");
}
```

> `result.Should().BeTrue` *se lit et se comprend facilement (le résultat devrait retourner 'vrai')*

Exemple 2 : Cosmos

Pour le projet cosmos, un DSL a été mis en place pour facilement tester certains aspects du langage :

```
[Fact]
public void TestDifferentNumber()
{
    TestBoolean("5".IsDifferentThan("6"), true);
}
```

> *Sans connaître le langage Cosmos, on comprend que ce teste vérifie la différence entre deux valeurs... Derrière les décors, IsDifferentThan convertit cela en langage Cosmos...*

Conclusion 🔭

Les extensions du langage sont une bonne manière d'accélérer le développement en enrichissant le langage de base avec ce qu'on a besoin de manière répétitive... Bien entendu, cela implique d'avoir une dépendance supplémentaire vers ces extensions et cet aspect doit être pris en compte dans la balance au moment de choisir de les utiliser ou pas...