**[C#] Comprendre les expressions lambda de C# 3**

**http://blogs.developpeur.org/tom/archive/2006/06/18/21672.aspx**

Avec l'arrivée de **C# 3**, on voit ce langage s'enrichir de nouvelles fonctionnalités Smile [:)]

L'une d'entre elles s'appelle les **expressions lambda**: derrière ce terme un peu barbare se cache un concept puissant que je vais tâcher de vous résumer en quelques lignes.

Avant de rentrer dans le vif du sujet, il faut se rappeler d'une des fonctionnalités offerte par C# 2: les méthodes anonymes.

Ces méthodes permettaient aux développeurs d’écrire des blocs de code en ligne, lorsque des délégués étaient nécessaires.

Par exemple, on pouvait tout à fait écrire quelque chose comme ceci :

List<int> list = new List<int>(new int [] { -1, 2, -5, 45, 5 });

List<int>positiveNumbers = list.FindAll(delegate(int i) { return i > 0;});

Les expressions lambdas représente en fait un moyen plus simple d’écrire les expressions anonymes.

Une expression lambda est composée de trois parties :

* Une liste de paramètre
* Le signe **=>**
* Une expression

Voici un exemple d’expression lambda :

(int i) => i % 2 == 0

Au niveau de la liste de paramètres, il y a deux choses à savoir :

* Les paramètres peuvent être typés explicitement ou implicitement. Dans le second cas, le type est déduit de l’expression qui constitue votre expression.
* S’il n’y a qu’un seul paramètre typé implicitement, les parenthèses sont facultatives

Voyons un peu comment nous pourrions utiliser cela dans notre code :

List<int> list = new List<int>(new int [] { -1, 2, -5, 45, 5 });

var positiveNumbers = list.FindAll((int i) => i > 0);

foreach(int positiveNumber in positiveNumbers)

{

Console.WriteLine(positiveNumber);

}

Le code de cette application, utilisant une expression lambda, fournit le même résultat que la version utilisant les méthodes anonymes : il est simplement plus clair à lire.

Bien entendu, rien ne vous empêche d’utiliser plusieurs paramètres dans votre expression lambda.

Commencez par créer une petite méthode d’extension que nous utiliserons plus tard. N’oubliez qu’une méthode d’extension est une méthode statiques. Pour bien faire, créer cette méthode dans une nouvelle classe, afin d’organiser au mieux votre code.

public static class FilterClass

{

public delegate bool KeyValueFilter<K, V>(K key, V value);

public static Dictionary<K, V> FilterBy<K, V>(this Dictionary<K, V>items, KeyValueFilter<K, V> filter)

{

var result = new Dictionary<K, V>();

foreach (KeyValuePair<K, V> element in items)

{

if (filter(element.Key, element.Value))

result.Add(element.Key, element.Value);

}

return result;

}

}

Ensuite, il ne vous reste plus qu’à construire votre expression lambda comme précédemment mais en modifiant votre expression pour travailler avec plusieurs paramètres :

// Expression lambda avec plusieurs paramètres

var listLanguages = new Dictionary<string, int>();

listLanguages.Add("C#", 5);

listLanguages.Add("VB", 3);

listLanguages.Add("Java", -5);

var bestLanguages = listLanguages.FilterBy((string name, int score) => name.Length == 2 && score > 0);

foreach(KeyValuePair<string, int>language in bestLanguages)

{

Console.WriteLine(language.Key);

}

Ici, on travaille bien sur une expression lambda qui possède plusieurs paramètres.