Les expressions lambda en C#

Hyacinthe MENIET

3 août 2019

Les expressions lambda sont, sans doute, l'une des fonctionnalités les plus puissantes livrées avec C# - songez qu'elles sont à la base de LINQ - mais en même temps, tant qu'on ne les a pas correctement assimilées, elles peuvent s'apparenter à une bizarrerie du langage. Dans cet article je vais introduire les concepts fondamentaux derrières les expressions lambda, disserter sur leurs intérêts et enfin montrer - à travers un exemple - comment les utiliser.

1. Pré-requis

— Vous êtes familier de C# 7 (.NET Framework 4.6.2 minimum) et de sa syntaxe.
— Vous avez installé et êtes familier de Microsoft Visual Studio (Community 2017 minimum). Le composant requis pour ce tutoriel est **Développement .NET Desktop**

2. Vue d'ensemble

2.1 Introduction aux expressions lambda

Commençons par une définition simple des expressions lambda : Une expression lambda est un morceau de code. Elle peut être assimilée à une méthode car elle accepte des paramètres et peut retourner une valeur. Mais vous pouvez également voir une expression lambda comme du code écrit à la volée et capable de capturer ou modifier des variables de son contexte.
Les expressions lambda sont particulièrement utiles lorsque vous souhaitez appliquer un traitement métier sur un objet ou une collection d'objets sans pour autant altérer le code de ces objets.

2.2. Déclaration et utilisation des expressions lambda

Une expression lambda a la forme suivante :

```
(parameters) => expression-or-statement-block
```

Notez que les parenthèses ne sont indispensables que si vous avez plusieurs paramètres.
Commençons par la déclaration d'une simple expression lambda :

```
String log = "";
List<int> ints = new List<int> { 0, 10, 20, 30 };
ints.ForEach(item => log += item);
Console.WriteLine(log);
```

L'expression lambda `(item => log += item)` concatène les éléments de la collection passée en paramètre. Pour ce faire, elle utilise la variable **log** déclarée dans son contexte. Après compilation, ce code affichera :

```
0102030
```

Vous pouvez également déclarer une expression lambda comme une fonction et assigner son résultat à une variable :

```
Func<string, string, int> sumLength = (f, s) => { return f.Length
    + s.Length; };
int l = sumLength("Hello","World");
Console.WriteLine(l);
```

Dans mon exemple ci-dessus, **int** représente le type de retour de l'expression lambda - qui peut être **void** - et les autres paramètres de **Func** (i.e. **string**) sont les types des paramètres de l'expression lambda.
Après compilation, ce code affichera :

```
10
```

2.3 Vue d'ensemble de l'article

Pour illustrer cet article, je vais me glisser dans les habits d'un infographiste. Afin de construire des formes évoluées, je dispose de 3 formes de bases : Triangle équilatéral, Carré et Cercle. En combinant ces formes de bases, je serai en mesure de dessiner des monstres, des humanoïdes, des décors etc.
Dans le code qui va suivre, chaque forme de base disposera d'une méthode retournant son aire. Cette méthode sera très utile, car en sommant les aires individuelles des formes de base qui composent ma forme évoluée, j'aurai une assez bonne approximation de l'aire couverte par cette dernière.
L'objet de cet article est donc d'utiliser les expressions lambda pour calculer la somme des aires des formes de base qui composent ma forme évoluée. Vous pouvez télécharger le code source du projet d'expressions lamba pour Visual Studio Community 2017.

3. Les structures de données

Dans Visual Studio, créez un projet *Application Console .NET Framework* nommé **LambdaExpression**. Cliquez droit sur votre projet et allez dans Propriétés. Là, remplacez l'espace de nom par défaut par **Dotmyself**.
Dans votre projet, ajoutez la classe abstraite Shape.cs contenant la méthode abstraire Area() :

```
using System;
namespace Dotmyself
{
/// <summary>
```

```
/// An abstract class that host the common features of all shapes
    .
/// Author : Hyacinthe MENIET
/// </summary>
abstract class Shape
{
public String Name;
public Shape(String name)
{
Name = name;
}
public abstract double Area();
}
}
```

Ajoutez la classe Triangle.cs qui spécialise la classe Shape :

```
using System;
namespace Dotmyself
{
/// <summary>
/// A derived class of Shape : a polygon with three corners and
    three edges.
/// Author : Hyacinthe MENIET
/// </summary>
class Triangle : Shape
{
public double Edge;
public Triangle(String name, double edge)
: base(name)
{
Edge = edge;
}
public override double Area()
{
return (Edge * Edge * Math.Sqrt(3)) / 4;
}
}
}
```

Ajoutez la classe Square.cs qui spécialise la classe Shape :

```
using System;
namespace Dotmyself
{
/// <summary>
/// A derived class of Shape, it has four equal sides and four
    equal angles.
/// Author : Hyacinthe MENIET
```

```
/// </summary>
class Square : Shape
{
public double Width;
public Square(String name, double width)
: base(name)
{
Width = width;
}
public override double Area()
{
return Width * Width;
}
}
}
```

Ajoutez la classe Circle.cs qui spécialise la classe Shape :

```
using System;
namespace Dotmyself
{
class Circle : Shape
{
/// <summary>
/// A derived class of Shape, consisting of those points in a
   plane which are equidistant from a given
/// point called the center.
/// Author : Hyacinthe MENIET
/// </summary>
public double Radius;
public Circle(String name, double radius)
: base(name)
{
Radius = radius;
}
public override double Area()
{
return Math.PI * Radius * Radius;
}
}
}
```

4. Programme principal

Ecrasez le contenu du fichier Program.cs par celui-ci :

```
using System;
using System.Collections.Generic;
```

```
namespace Dotmyself
{
/// <summary>
/// It computes the area covered by all the shapes
/// Author : Hyacinthe MENIET
/// </summary>
class Program
{
static void Main(string[] args)
{
List<Shape> shapes = new List<Shape>
{
new Triangle("T1",2.5),
new Square("S1",4.75),
new Triangle("T2",1.66),
new Circle("C1",3.25)
};
double total = 0;
shapes.ForEach(
s =>
{
Console.WriteLine("The individual area of {0} is {1} units", s.
   Name, s.Area());
total += s.Area();
});
Console.WriteLine("The shapes in this drawing cover an area of
   {0} units.", total);
Console.ReadLine();
}
}
}
```

L'exécution de ce programme devrait vous afficher :

```
The individual area of T1 is 2,70632938682637 units
The individual area of S1 is 22,5625 units
The individual area of T2 is 1,1932098013342 units
The individual area of C1 is 33,1830724035422 units
The shapes in this drawing cover an area of 59,6451115917028
   units.
```