Instituto Superior de Engenharia de Lisboa
Guião para laboratório de Ambientes Virtuais de Execução / LEIC
5 de Maio de 2016

Objectivos: Prática com delegates, genéricos e iteradores. Análise e definição de iteradores com construção yield.

Prática I

1. Usando o *delegate* `System.Action<T>` e o método `ForEach` da classe `System.Array`, implemente o método `PrettyPrint` segundo as indicações nas alíneas a) e b). Para cada elemento do *array*, o método apresenta na consola o nome do tipo e a representação em *string*:

   ```
   static void PrettyPrint<T>(T[] values)
   ```

   a) Com *delegates* instanciados explicitamente.

   b) Com funções lambda.

2. Considere o seguinte excerto de código:

```
public struct Point{ /*...*/ }

public class Program{

   public static Point[] ConvertingPoints( int[] array ){
      //...
   }

   public static void Main(){
      int[] values={1,2,3,4,5,6,7,8,9};
      Point[] newValues=ConvertingPoints(values );
      foreach(Point j in newValues)
         Console.WriteLine(j);
   }
}
```

Output:

```
(1,1)
(2,4)
(3,9)
(4,16)
(5,25)
(6,36)
(7,49)
(8,64)
(9,81)
```

Considere também o *delegate*

`public delegate TOutput Converter<TInput,TOutput> (TInput input)`

Definido no espaço de nomes `System`, e o método

`public static U[] ConvertAll<T,U>(T[] array, Converter<T,U> converter)`

da classe `System.Array`.

   a) Tirando partido do método `ConvertAll`, implemente o método `ConvertingPoints` de modo a que que produza o *output* pretendido, acrescentando a `Point` o que entender necessário.

3. Na plataforma .NET a interface `IEnumerable<T>` representa sequências de T e a interface `IEnumerator<T>` representa iteradores para essas sequências.

   i. Implemente os métodos seguintes retornado um iterador lazy, ou seja, um iterador que determina qual o elemento *n* da sequência depois de determinar e retornar o *n-1*.

   ii. Realize um programa de teste para cada alínea.

a)
`static IEnumerable<int> BetweenRange(int begin, int end)`
que retorna a sequência de inteiros compreendidos em *[begin, end]*.

b)
`static IEnumerable<T> Concat<T>(IEnumerable<T> seq1, IEnumerable<T> seq2)`
que retorna a sequência que representa a concatenação das sequências seq1 e seq2.

Análise

1. Defina o *delegate*
   `delegate int Operation(String s)`
Usando a ferramenta ildasm, analise o *assembly* produzido. O método Invoke da classe Operation está vazio. É gerado por quem?

2. Considere o seguinte código Program.cs. Usando a ferramenta `ildasm` identifique:
   a. A construção `yield` tem suporte directo em alguma instrução intermédia?
   b. Qual o nome do tipo concreto de objecto retornado pelo método `Count123`?
   c. A construção `foreach` obtém um `IEnumerator<int>` a partir do `IEnumerable<int>` retornado por Count123. Qual o nome do tipo concreto de objecto retornado pelo método `GetEnumerator`?
   d. Os métodos `Count123` e `Count1ToN` retornam o mesmo tipo concreto?
   e. No tipo concreto retornado pelo método `Count123` existem dois campos que parecem estar relacionados com a manutenção de estado do iterador: `state` e `current`.
      i. Altere o método para retornar outro tipo de sequência (por exemplo, `IEnumerable<string>`, `IEnumerable<DateTime>`) e determine qual destes campos mantém o valor corrente do iterador.
      ii. De que outra forma poderia chegar à mesma conclusão?

Mais informação sobre este tema poder obtida em
http://csharpindepth.com/Articles/Chapter6/IteratorBlockImplementation.aspx.

Prática II

1. Usando a construção yield:
   a. Reescreva as alíneas I.3.a e I.3.b) para retornarem um iterador *lazy*.

   b. `IEnumerable<R> Zip<TA, TB, R>(IEnumerable<TA> a, IEnumerable<TB> b, Func<TA, TB, R> join)`, que retorna uma sequência de elementos resultantes da aplicação da função `join` aos elementos de a e b. A sequência resultante termina assim que uma das sequências não tenha mais elementos.