Instituto Superior de Engenharia de Lisboa

Prompt, 2011

Sessão 5

Objectivos:

- Prática de utilização e criação de métodos genéricos
- Compreensão básica do suporte a iteradores na linguagem C#

Parte I

Para os exercícios 2 e 3 implemente o respectivo código de teste na forma de testes unitários.

1. Seja a classe `RationalNumber`:

```
public struct RationalNumber
{
    private readonly int  _numerator, _denominator;

    RationalNumber(int numerator, int denominator)
    {
        if (denominator == 0) throw new InvalidRationalException();
        _numerator = numerator;
        _denominator = denominator;
    }

    public static RationalNumber operator+(RationalNumber n1, RationalNumber n2)
    {
        int nn, nd;

        if (n1._denominator == n2._denominator )
        {
            nn = n1._numerator + n2._numerator;
            nd = n1._denominator;
        }
        else
        {
            nn = n1._numerator*n2._denominator + n2._numerator*n1._denominator;
            nd = n1._denominator*n2._denominator;
        }
        return new RationalNumber( nn, nd);
    }

    public override string ToString()
    {
        var sb = new StringBuilder();
        sb.Append(_numerator);
        sb.Append('/');
        sb.Append(_denominator);
        return sb.ToString();
    }
}
```

Altere a classe por forma a suportar a sua utilização eficiente no seguinte programa:

```
class Program {
  static void Main(string[] args) {

      List<RationalNumber> rnumbers = new List<RationalNumber>
                                          {
                                                 new RationalNumber(1, 1),
                                                 new RationalNumber(2, 1),

                                          };

      var n = rnumbers.Where(r => r.Equals(new RationalNumber(1, 1)));

      foreach(var r in n)
          Console.WriteLine(r);

      rnumbers.Sort();
      foreach (var r in n)
          Console.WriteLine(r);

  }
```

2. Realize o método de extensão:

```
public static IEnumerable<T> RemoveRepeated<T>(IEnumerable<T> seq)
```

que retorna uma sequência da qual foram removidos os elementos da sequência original.

3. Altere o método de extensão `OrderBy mostrado abaixo` e realize o método `ThenBy`, por forma a que ao resultado da função `OrderBy` possa ser aplicada a função `ThenBy`. Minimize o número de ordenações realizadas. Faça um programa de teste com uma colecção de instâncias do tipo `Point` também presente no código em anexo.

```
public static IEnumerable<T> OrderBy<T,U>(this IEnumerable<T> seq, Func<T,U> criterium)
            where  U : IComparable<U>
{
    var a = seq.ToArray();
    Array.Sort(a, (t1, t2) => criterium(t1).CompareTo(criterium(t2)));
    return a;
}
```

Parte II