08

SortedSet

Transcrição

Neste vídeo, lidaremos com um novo projeto *console application*. Veremos um novo tipo de coleção, só que, antes, trabalharemos com a coleção de conjuntos que vimos na Parte 1 do curso de C# Collections, que é a coleção *HashSet*.

Criaremos um conjunto de alunos, que armazenará strings, ou seja, um HashSet de strings.

Declararemos a interface `ISet<string>` , e o nome da coleção será `alunos` .

Adicionaremos os alunos, e veremos o que acontece quando adicionamos um dos nomes em letra maiúscula.

```
static void Main(string[] args)
{
        //Conjunto de alunos:
        ISet<string> alunos
            = new HashSet<string>()
            {
                "Vanessa Tonini",
                "Ana Losnak",
                "Rafael Nercessian",
                "Priscila Stuani"
            };

            //adicionar: Rafael Rollo
            alunos.Add("Rafael Rollo");
            //adicionar: Fabio Gushiken
            alunos.Add("Fabio Gushiken");
            //adicionar: FABIO GUSHIKEN
            alunos.Add("FABIO GUSHIKEN");

            foreach (var aluno in alunos)
            {
                    Console.WriteLine(aluno);
            }
```

Com o comando "Ctrl + F5", executaremos o programa.

Surgirá uma caixa de texto onde veremos a listagem dos alunos. No caso do nome minúsculo e maiúsculo, o programa não identificou como sendo o mesmo nome. Veremos adiante, como podemos evitar isso.

Por enquanto, não temos uma lista ordenada. Para isso, declararemos um `SortedSet` .

```
static void Main(string[] args)
{
        //Conjunto de alunos:
        ISet<string> alunos
            = new SortedSet<string>()
            {
                "Vanessa Tonini",
```

```
            "Ana Losnak",
            "Rafael Nercessian",
            "Priscila Stuani"
        };

        //adicionar: Rafael Rollo
        alunos.Add("Rafael Rollo");
        //adicionar: Fabio Gushiken
        alunos.Add("Fabio Gushiken");
        //adicionar: FABIO GUSHIKEN
        alunos.Add("FABIO GUSHIKEN");

        foreach (var aluno in aluno)
        {
                Console.WriteLine(aluno);
        }
```

Utilizaremos o comando "Ctrl + F5" para executarmos o programa.

Como resultado, temos os nomes ordenados.

O próximo passo, será fazer com que o sistema identifique os nomes em duplicidade, sem distinção de caixa alta ou baixa.

No construtor do `SortedSet`, temos uma sobrecarga com um parâmetro que é um comparador, o `IComparer`.

Ele deverá ser uma classe, que deverá identificar quando houver nomes iguais, não importando se estão em maiúscula ou minúscula.

Criaremos uma nova classe, que chamaremos de `ComparadorMinusculo`. Com o cursor sobre seu nome, utilizaremos o atalho "Ctrl + .", e selecionaremos a opção "*Generate Class ComparadorMinusculo".

Feito isso, ao final do código, teremos:

```
}

internal class ComparadorMinusculo : IComparer<string>
{
```

Esta classe implementa a interface `IComparer`, utilizando o comando "Ctrl + .", selecionaremos a opção "*implement interface*".

Utilizaremos novamente o atalho "Ctrl + .", e assim serão criados os métodos necessários para implementar esta interface, que é o método *compare*.

Com isso, ele irá comparar a string "x", com a string "y", e dirá se elas são iguais.

Como queremos igualar os dois tipos de strings, maiúsculos e minúsculos, colocaremos um retorno, que fará a comparação do "x" e "y", ignorando maiúsculas e minúsculas. Para isso, declararemos `InvariantCultureIgnoreCase`.

Ao final, teremos a classe da seguinte forma:

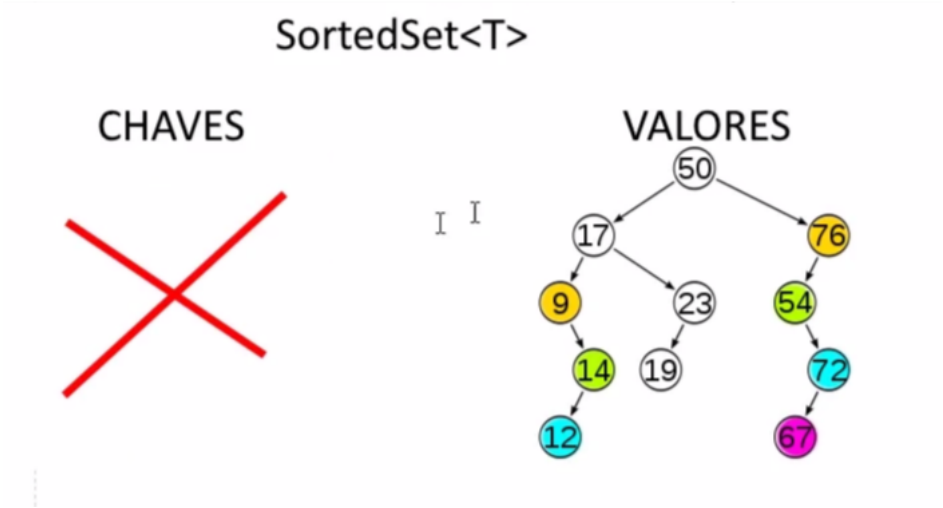
```
internal class ComparadorMinusculo : IComparer<string>
{
        public int Compare(string x, string y)
        {
                return string.Compare(x, y, StringComparison.InvariantCultureIgnoreCase)
        }
```

Feito isso, executaremos a aplicação, utilizando o atalho "Ctrl + F5".

Desta forma, teremos os nomes listados, em ordem alfabética.

O nome "Fabio", que havíamos inserido em minúsculo e maiúsculo, foi escrito somente uma vez. Isso porque o programa identificou que os nomes são idênticos, com a única diferença da segunda entrada ser em caixa alta.

A implementação interna de um `SortedSet` funciona da seguinte forma:



Aqui, temos uma árvore binária somente na parte de valores, assim como no `SortedDictionary`.

O dicionário não contém chaves, ou seja, não é uma coleção associativa. Os valores, são as próprias chaves dos elementos de um `SortedSet`.

Internamente, ele utiliza uma árvore binária, que é balanceada, para poder armazenar e saber a ordem correta dos elementos.

Veremos agora algumas operações que podem ser feitas com o `SortedSet`, métodos que serão úteis para trabalharmos com o conceito de conjuntos matemáticos.

Declararemos um novo conjunto:

```
ISet<string> outroConjunto = new HashSet<string>();
```

Faremos em seguida um método para descobrirmos se um conjunto é subconjunto de outro:

```
ISet<string> outroConjunto = new HashSet<string>();

//este conjunto é subconjunto de outro? IsSubsetOf
alunos.IsSubsetOf(outroConjunto);
```

Veremos também se nosso conjunto é um superconjunto de outro, ou seja, se contém algum outro:

```
ISet<string> outroConjunto = new HashSet<string>();

//este conjunto é subconjunto de outro? IsSubsetOf
alunos.IsSubsetOf(outroConjunto);

//este conjunto é superconjunto de outro? IsSupersetOf
alunos.IsSupersetOf(outroConjunto);
```

Para saber se os conjuntos contêm os mesmos elementos, usaremos:

```
ISet<string> outroConjunto = new HashSet<string>();

//este conjunto é subconjunto de outro? IsSubsetOf
alunos.IsSubsetOf(outroConjunto);

//este conjunto é superconjunto de outro? IsSupersetOf
alunos.IsSupersetOf(outroConjunto);

//os conjuntos contêm os mesmo elementos? SetEquals
alunos.SetEquals(outroConjunto);
```

Para subtrair os elementos de outra coleção, que também estão neste conjunto, utilizaremos:

```
ISet<string> outroConjunto = new HashSet<string>();

//este conjunto é subconjunto de outro? IsSubsetOf
alunos.IsSubsetOf(outroConjunto);

//este conjunto é superconjunto de outro? IsSupersetOf
alunos.IsSupersetOf(outroConjunto);

//os conjuntos contêm os mesmo elementos? SetEquals
alunos.SetEquals(outroConjunto);

//subtrai os elementos da outra coleção que também estão nesse? ExceptWith
alunos.ExceptWith(outroConjunto);
```

Para sabermos quais são os pontos de intersecção entre os conjuntos:

```
ISet<string> outroConjunto = new HashSet<string>();

//este conjunto é subconjunto de outro? IsSubsetOf
alunos.IsSubsetOf(outroConjunto);

//este conjunto é superconjunto de outro? IsSupersetOf
alunos.IsSupersetOf(outroConjunto);

//os conjuntos contêm os mesmo elementos? SetEquals
alunos.SetEquals(outroConjunto);
```

```
//subtrai os elementos da outra coleção que também estão nesse? ExceptWith
alunos.ExceptWith(outroConjunto);

//intersecção dos conjuntos - IntersectWith
alunos.IntersectWith(outroConjunto);
```

Para sabermos quais os elementos que estão em um, ou outro conjunto, eliminando toda a parte de intersecção, utilizamos:

```
ISet<string> outroConjunto = new HashSet<string>();

//este conjunto é subconjunto de outro? IsSubsetOf
alunos.IsSubsetOf(outroConjunto);

//este conjunto é superconjunto de outro? IsSupersetOf
alunos.IsSupersetOf(outroConjunto);

//os conjuntos contêm os mesmo elementos? SetEquals
alunos.SetEquals(outroConjunto);

//subtrai os elementos da outra coleção que também estão nesse? ExceptWith
alunos.ExceptWith(outroConjunto);

//intersecção dos conjuntos - IntersectWith
alunos.IntersectWith(outroConjunto);

//somente em um ou outro conjunto - SymmetricExceptWith
alunos.SymmetricExceptWith(outroConjunto);
```

Por fim, para sabermos qual é a união entre os conjuntos, utilizaremos:

```
ISet<string> outroConjunto = new HashSet<string>();

//este conjunto é subconjunto de outro? IsSubsetOf
alunos.IsSubsetOf(outroConjunto);

//este conjunto é superconjunto de outro? IsSupersetOf
alunos.IsSupersetOf(outroConjunto);

//os conjuntos contêm os mesmo elementos? SetEquals
alunos.SetEquals(outroConjunto);

//subtrai os elementos da outra coleção que também estão nesse? ExceptWith
alunos.ExceptWith(outroConjunto);

//intersecção dos conjuntos - IntersectWith
alunos.IntersectWith(outroConjunto);

//somente em um ou outro conjunto - SymmetricExceptWith
alunos.SymmetricExceptWith(outroConjunto;

//união de conjuntos - UnionWith
alunos.UnionWith(outroConjunto);
```

Com isso, fechamos o capítulo sobre coleções ordenadas, e passaremos a trabalhar com arrays multidimensionais.