Listas e dicionários

Listas

- É muito comum trabalhar com listas de objetos, e, no passado tratar listas de objetos dava muito trabalho.
- A linguagem C# disponibiliza muitos recursos que faz com que o tratamento de listas de objetos se tornasse algo muito simples.

Listas

- Vamos imaginar que você tenha uma classe Pessoa definida com o seguinte código:
- Note que temos dois construtores na classe Pessoa.

```
public class Pessoa
{
    public Pessoa() { }

    public string Nome { get; set; }
    public int Idade { get; set; }

    public Pessoa(int idade, string nome)
    {
        this.Idade = idade;
        this.Nome = nome;
    }
}
```

Listas

```
class Program
{
    static List<Pessoa> pessoas;

    static void Main(string[] args)
    {
        pessoas = new List<Pessoa>();
        pessoas.Add(new Pessoa(40, "Ricardo"));
        pessoas.Add(new Pessoa(25, "Jefferson"));
        pessoas.Add(new Pessoa(45, "Miriam"));
        foreach (Pessoa p in pessoas)
        {
            Console.WriteLine(p.Nome + " " + p.Idade);
        }
        Console.ReadKey();
    }
}
```

Listas

- Usamos a classe **List**<T> que representa uma lista fortemente tipada de objetos do tipo Pessoa que podem ser acessados pelo seu índice, e, usei o método **Add**() para incluir objetos no fim da lista.

- A classe **List**<T> também fornece uma série de métodos que vão facilitar muito sua vida. Dentre eles eu destaco os métodos : **ForEach**, **FindAll**, **Find** e **Sort**.

- **ForEach** nos permite acessar cada item na lista iterando sobre ele;
- **FindAll** permite procurar por objetos na lista que correspondem a uma condição específica;
- **Find** permite procurar por um elemento na lista que corresponde a uma condição específica retornando a primeira ocorrência;
- **Sort** permite classificar os objetos da lista;

Lista não ordenada

- Para obter lista não ordenada basta percorrer a lista usando um **ForEach**.

```
static void ListaNaoOrdenada()
{
    Console.WriteLine("Lista não ordenada");
    pessoas.ForEach(delegate (Pessoa p)
    {
        Console.WriteLine(p.Idade + " " + p.Nome);
    });
}
```

Lista ordenada por nome

- Aqui o método **Sort()** classifica os elementos em toda lista usando o comparador padrão.

```
static void ListaOrdenadaPorNome()
{
    Console.WriteLine("Lista Ordenada por Nome");
    pessoas.Sort(delegate (Pessoa p1, Pessoa p2)
    {
        return p1.Nome.CompareTo(p2.Nome);
    });
    pessoas.ForEach(delegate (Pessoa p)
    {
        Console.WriteLine(p.Idade + " " + p.Nome);
    });
}
```

Lista ordenada por idade

- Aqui o método **Sort**() classifica os elementos em toda lista usando o comparador padrão.

```
static void ListaOrdenadaPorIdade()
{
    Console.WriteLine("Lista Ordenada por Idade");
    pessoas.Sort(delegate (Pessoa p1, Pessoa p2)
    {
        return p1.Idade.CompareTo(p2.Idade);
    });
    pessoas.ForEach(delegate (Pessoa p)
    {
        Console.WriteLine(p.Idade + " " + p.Nome);
    });
}
```

Inserir um item na lista em uma posição do índice

- O método **Insert**() insere o elemento na lista em uma posição especificada.

```
static void ListaInserirItemNaPosicao()
{
    Console.WriteLine("Inserindo uma pessoa na posição 1 e outra na posição 3");

    pessoas.Insert(1, new Pessoa() { Nome = "Bob Dylan", Idade = 78 });
    pessoas.Insert(3, new Pessoa() { Nome = "Jimmi Page", Idade = 81 });
}
```

Converter a lista para um Array

- O método **ToArray**() copia os elementos da **List**<T> para um novo array.

```
static void ListaConverterParaArray()
{
    Console.WriteLine("Convertendo a lista para um Array");
    Pessoa[] ListaArray = pessoas.ToArray();
    foreach (Pessoa p in ListaArray)
    {
        Console.WriteLine(p.Nome + " " + p.Idade);
    }
}
```

Localizar a pessoa mais jovem na lista

- O método **FindAll** recupera todos os elementos que combinam as condições definidas pelo predicado especificado.

```
static void ListaLocalizaPessoaMaisJovem()
{
    List<Pessoa> jovem = pessoas.FindAll(delegate (Pessoa p) { return p.Idade < 45; });

    Console.WriteLine("Idade é menor que 25 : ");
    jovem.ForEach(delegate (Pessoa p)
    {
        Console.WriteLine(p.Nome + " " + p.Idade);
    });
}
```

Dicionários

Dicionário

- A classe **Dictionary** representa uma coleção de chaves e valores.
- É uma coleção do tipo chave/valor e implementa a interface **IDictionary** que possui duas coleções no seu interior uma para guardar a chave e outra para guardar o valor.

Dicionário

- Esta classe esta definida no namespace System.Collections.Generic sendo uma classe genérica e pode armazenar qualquer tipo de dados em uma forma de chaves e valor, onde cada chave deve ser exclusiva na coleção.

- A classe **Dictionary** fornece recursos semelhantes a uma **Hashtable**, mas é fortemente tipada.
- Isso significa que seu código não precisa converter de objetos genéricos em tipos específicos. Isso também significa que a classe **Dictionary** garante que seu código passe os tipos corretos de objetos para ele.

Dicionário

- O objeto **Dictionary** pode ser atribuído a uma variável do tipo IDictionary<Tkey,TValue> ou à classe Dictionary <TKey,Tvalue>. Exemplo de inicialização:

- IDictionary<int, string> dict = new Dictionary<int, string>();
-
- Dictionary<int, string> dict = new Dictionary<int, string>();
- No código cima especificamos os tipos de chave e valor ao declarar um objeto de dicionário.
- Um int é um tipo de chave e string é um tipo de valor que será armazenado em um objeto de dicionário chamado dict. Você pode usar qualquer tipo de dados C# válido para chaves e valores.

Dicionário

- As principais propridades da classe Dictionary são:
  - **Count** - Obtém o número total de elementos no Dictionary<TKey,TValue>.
  - **IsReadOnly** - Retorna um booleano indicando se o Dictionary<TKey,TValue> é somente leitura.
  - **Item** - Obtém ou define o elemento com a chave especificada no Dictionary<TKey,TValue>.
  - **Keys** - Retorna a coleção de chaves do Dictionary<TKey,TValue>
  - **Values** - Retorna a coleta de valores no Dictionary<TKey,TValue>

Dicionário

- Os principais métodos da classe Dictionary são:
  - Add - Adiciona um item à coleção Dictionary.
  - Add - Adiciona pares de valores-chave na coleção Dictionary<TKey,TValue>.
  - Remove - Remove a primeira ocorrência do item especificado do Dictionary<TKey,TValue>.
  - Remove - Remove o elemento com a chave especificada.
  - ContainsKey - Verifica se a chave especificada existe em Dictionary<TKey,TValue>.
  - ContainsValue - Verifica se o valor especificado existe em Dictionary<TKey,TValue>.
  - Clear - Remove todos os elementos do Dictionary<TKey,TValue>.
  - TryGetValue - Retorna true e atribui o valor com a chave especificada, se a chave não existir, retorna false.

Dicionário - Adicionando elementos a um Dictionary

- Use o método Add() para adicionar um par chave-valor ao dicionário. Add(Tkey,TValue)

```
static void Main(string[] args)
{
    IDictionary<int, string> dic1 = new Dictionary<int, string>();
    dic1.Add(1, "Maria");
    dic1.Add(2, "Paulo");
    dic1.Add(3, "Pedro");
}
```

Dicionário - Adicionando elementos a um Dictionary

- Podemos inicializar um Dicionario usando a sintaxe do inicializador de coleções com chaves e valores, conforme mostrado abaixo.

```
static void Main(string[] args)
{
    IDictionary<int, string> dic2 = new Dictionary<int, string>()
    {
                    {1,"Maria"},
                    {2, "Paulo"},
                    {3,"Pedro"}
    };

}
```

Dicionário - Acessando elementos de um Dictionary

- Os elementos do dicionário podem ser acessados de muitas maneiras quer usando um laço foreach ou um indexador.

- Usamos um foreach ou loop para iterar sobre todos os elementos do dicionário. O dicionário armazena pares de valores-chave. Assim, você pode usar um tipo KeyValuePair<TKey,TValue> ou uma variável implicitamente tipada no laço foreach, conforme mostrado abaixo.

Dicionário - Acessando elementos de um Dictionary

```
static void Main(string[] args)
{
    Dictionary<int, string> dic1 = new Dictionary<int, string>()
    {
                                        {1,"Banana"},
                                        {2, "Laranja"},
                                        {3, "Manga"},
                                        {4, "Abacate"},
                                        {5,"Maça"}
    };
    foreach (KeyValuePair<int, string> item in dic1)
    {
        Console.WriteLine(item.Key+" "+item.Value);
    }
}
```

Dicionário - Acessando elementos de um Dictionary

- Podemos também usar o dicionario como um array para acessar seus elementos individuais. Para isso basta especificar a chave (não o índice) para obter um valor de um dicionário usando o indexador como um array.

```
static void Main(string[] args)
{
    Dictionary<int, string> dic1 = new Dictionary<int, string>()
    {
                                        {1,"Banana"},
                                        {2, "Laranja"},
                                        {3, "Manga"},
                                        {4, "Abacate"},
                                        {5,"Maça"}
    };
    Console.WriteLine(dic1[2]);   // retorna laranja
    Console.WriteLine(dic1[4]);   // retorna abacate
    Console.ReadKey();
}
```

Dicionário - Acessando elementos de um Dictionary

- Se você não tiver certeza sobre a chave, use o método TryGetValue() que vai retornar false se não puder encontrar chaves em vez de gerar uma exceção.

```
static void Main(string[] args)
{
    Dictionary<int, string> dic1 = new Dictionary<int, string>()
    {
                                        {1,"Banana"},
                                        {2, "Laranja"},
                                        {3, "Manga"},
                                        {4, "Abacate"},
                                        {5,"Maça"}
    };
    string resultado;
    if (dic1.TryGetValue(4, out resultado))
    {
        Console.WriteLine(resultado);
    }
    else
    {
        Console.WriteLine("Não foi possível achar a chave especificada.");
    }
}
```

Dicionário - Acessando elementos de um Dictionary

- Se você não tiver certeza sobre a chave, use o método TryGetValue() que vai retornar false se não puder encontrar chaves em vez de gerar uma exceção.

```
static void Main(string[] args)
{
    Dictionary<int, string> dic1 = new Dictionary<int, string>()
    {
                                        {1,"Banana"},
                                        {2, "Laranja"},
                                        {3, "Manga"},
                                        {4, "Abacate"},
                                        {5,"Maça"}
    };
    string resultado;
    if (dic1.TryGetValue(4, out resultado))
    {
        Console.WriteLine(resultado);
    }
    else
    {
        Console.WriteLine("Não foi possível achar a chave especificada.");
    }
}
```

Dicionário - Verificar se há elementos existentes

- Um Dicionário contém vários métodos para determinar se um ele contém elementos ou chaves especificados. Use o método ContainsKey() para verificar se uma chave especificada existe no dicionário ou não.

- Use o método Contains() para verificar se um par de chave e valor especificado existe no dicionário ou não.

- Assinaturas:

  - bool ContainsKey (TKey key)
  - bool Contains(item KeyValuePair<TKey,TValue>)

Dicionário - Verificar se há elementos existentes

```
static void Main(string[] args)
{
    Dictionary<int, string> dic1 = new Dictionary<int, string>()
    {
                                        {1,"Banana"},
                                        {2, "Laranja"},
                                        {3, "Manga"},
                                        {4, "Abacate"},
                                        {5,"Maça"}
    };
    Console.WriteLine(dic1.ContainsKey(1)); // retorna true
    Console.WriteLine(dic1.ContainsKey(6)); // retorna false


    Console.WriteLine(dic1.ContainsValue("Manga"));// retorna true

    Console.ReadKey();
    Console.ReadKey();
}
```

Dicionário - Remover elementos de um dicionário

- Use o método Remove() para remover um item existente do dicionário. Este método possui duas sobrecargas:

  - Um método aceita uma chave - bool Remove(Tkey key)
  - Outro método aceita um KeyValuePair<> como um parâmetro - bool Remove(KeyValuePair<TKey,TValue>)

Dicionário - Remover elementos de um dicionário

```
static void Main(string[] args)
{
    Dictionary<int, string> dic1 = new Dictionary<int, string>()
    {
                                        {1,"Banana"},
                                        {2, "Laranja"},
                                        {3, "Manga"},
                                        {4, "Abacate"},
                                        {5,"Maça"}
    };
    // remove o item com chave igual a 1
    dic1.Remove(1);
    Console.ReadKey();
}
```

Dicionário - Ordenando um dicionario

- Para ordenar um dicionário use a coleção genérica SortedDictionary que ordena o dicionário com base nas chaves.

```
static void Main(string[] args)
{
    //criando um dicionario ordenado
    SortedDictionary<string, int> dic2 = new SortedDictionary<string, int>();
    // Adicionando strings e chaves do tipo int
    dic2.Add("zebra", 5);
    dic2.Add("cachorro", 2);
    dic2.Add("gato", 9);
    dic2.Add("pardal", 4);
    dic2.Add("C#", 100);
    // Verifica se gato existe no dicionario
    if (dic2.ContainsKey("gato"))
    {
        Console.WriteLine("tem um gato ai...");
    }
    // Verifica se tem zebra
    if (dic2.ContainsKey("zebra"))
    {
        Console.WriteLine("Deu zebra pois não tem zebra ai...");
    }
    // Verifica se contém C#
    // e se tiver pega o valor
    int v;
    if (dic2.TryGetValue("C#", out v))
    {
        Console.WriteLine(v);
    }
    // Imprime o SortedDictionary em ordem alfabética
    foreach (KeyValuePair<string, int> p in dic2)
    {
        Console.WriteLine(p.Key+" "+p.Value);
    }
    Console.ReadKey();
}
```