Linguagem de Programação 3

Aula 10
Classes de Coleções

l.bertholdo@ifsp.edu.br

Conteúdo

- Classes de Coleção Genéricas
  - Dictionary <TKey, TValue>
  - HashSet <T>
- Outras Coleções Genéricas
- Classes de Coleção Não Genéricas

l.bertholdo@ifsp.edu.br

Dictionary<TKey, TValue>

- A classe **Dictionary<TKey, TValue>** é similiar a uma lista de objetos, com a diferença de que seus elementos não precisam ser indexados através de números inteiros.
- Nessa coleção, cada elemento é representado por meio de uma chave associativa, onde **TKey** é o tipo de dado da chave e **TValue** o tipo de dado do valor a ser armazenado.
- Esta coleção não mantém nenhuma ordem, nem permite chaves duplicadas, porém valores duplicados podem existir.
- Coleções Dictionary<TKey, TValue> são usadas quando se deseja identificar os elementos de uma coleção sem necessariamente ter que usar números inteiros.

l.bertholdo@ifsp.edu.br

Dictionary<TKey, TValue>

- Principais métodos:
  - **Add:** Adiciona um elemento à coleção.
    Sintaxe: <nome_coleção>.Add(<chave>, <valor>)
  - **Remove:** Remove um elemento da coleção por meio de sua chave. **Sintaxe:** <nome_coleção>.Remove(<chave>)
  - **ContainsKey:** Verifica se a coleção contém uma determinada chave. **Sintaxe:** <nome_coleção>.ContainsKey(<chave>)
  - **ContainsValue:** Verifica se a coleção contém um determinado valor. **Sintaxe:** <nome_coleção>.ContainsValue(<valor>)

* A classe Dictionary<TKey, TValue> pode ser inicializada com valores ao ser declarada. Exemplo:

```
Dictionary<string, int> nomes = new Dictionary<string, int>() { { "João", 25 }, { "Maria", 34 }, { "Pedro", 18 } };
```

l.bertholdo@ifsp.edu.br

Dictionary<TKey, TValue>

- Principais propriedades:
  - **Count:** Retorna a quantidade de elementos da coleção.
    Sintaxe: <nome_coleção>.Count
  - **Keys:** Retorna todas as chaves da coleção.
    Sintaxe: <nome_coleção>.Keys
  - **Values:** Retorna todos os valores da coleção.
    Sintaxe: <nome_coleção>.Values
  - **Key:** Retorna a chave de um elemento da coleção.
    Sintaxe: <elemento>.Key
  - **Value:** Retorna o valor de um elemento da coleção.
    Sintaxe: <elemento>.Value

l.bertholdo@ifsp.edu.br

Dictionary<TKey, TValue>

```
class Dicionario {
    static void Main(string[] args) {
        Dictionary<string, int> nomes = new Dictionary<string, int>();

        // Adiciona 5 elementos à coleção "nomes".
        // KeyValuePair<TKey, TValue> é uma struct que contém uma
        // cópia do par "chave/valor" de um elemento da coleção.
        foreach (KeyValuePair<string, int> n in new Dictionary<string, int>
        { { "João", 25 }, { "Maria", 34 }, { "Pedro", 18 }, { "Julia", 22 }, { "Carlos", 41 } })
            // Key: Recupera a chave do elemento. Value: Recupera o valor do elemento.
            nomes.Add(n.Key, n.Value);

        foreach (KeyValuePair<string, int> n in nomes) // Imprime a coleção.
            Console.WriteLine(n); Console.WriteLine();

        nomes.Remove("Maria"); // Remove o elemento cuja chave é "Maria".

        nomes["Pedro"] = 19; // Altera o valor da chave "Pedro" de 18 para 19.

        foreach (KeyValuePair<string, int> n in nomes) // Imprime a coleção.
            Console.WriteLine(n); Console.WriteLine();

        Console.ReadKey();
    }
}
```

```
[João, 25]
[Maria, 34]
[Pedro, 18]
[Julia, 22]
[Carlos, 41]

[João, 25]
[Pedro, 19]
[Julia, 22]
[Carlos, 41]
```

l.bertholdo@ifsp.edu.br

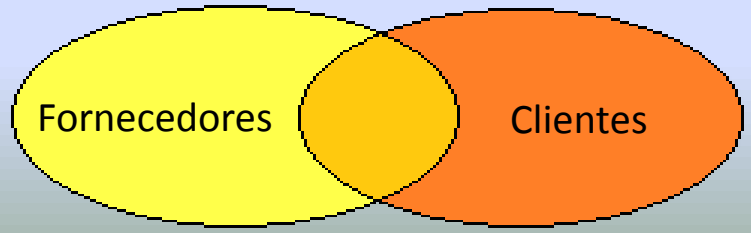
Dictionary<TKey, TValue>

```
class Dicionario {
    static void Main(string[] args) {
        Dictionary<string, int> nomes = new Dictionary<string, int>();

        // Adiciona 5 elementos à coleção "nomes".
        foreach (KeyValuePair<string, int> n in new Dictionary<string, int>
        { { "João", 25 }, { "Maria", 34 }, { "Pedro", 18 }, { "Julia", 22 }, { "Carlos", 41 } })
            nomes.Add(n.Key, n.Value);

        if (nomes.ContainsKey("Carlos")) // Se a coleção contém a chave "Carlos".
            Console.WriteLine("A idade de Carlos é: " + nomes["Carlos"]); // Imprime o valor do elemento.

        if (nomes.ContainsValue(22)) // Se a coleção contém o valor 22.
            Console.WriteLine("A coleção tem este valor.");

        Console.WriteLine("Quantidade de elementos da coleção: " + nomes.Count);

        Console.Write("Chaves da coleção: ");
        foreach (string k in nomes.Keys) // Retorna as chaves da coleção.
            Console.Write(k + ", "); Console.WriteLine();

        Console.Write("Valores da coleção: ");
        foreach (int v in nomes.Values) // Retorna os valores da coleção.
            Console.Write(v + ", ");
        Console.ReadKey();
    }
}
```

```
A idade de Carlos é: 41
A coleção tem este valor.
Quantidade de elementos da coleção: 5
Chaves da coleção: João, Maria, Pedro, Julia, Carlos,
Valores da coleção: 25, 34, 18, 22, 41,
```

HashSet<T>

- A classe HashSet<T> é usada para realizar operações com conjuntos de valores.
- Em uma coleção HashSet<T>, a posição de cada elemento é definida por uma chave codificada e exclusiva.
- Além de não manter nenhuma ordem, este tipo de coleção não permite elementos com valores duplicados.
- Coleções HashSet<T> são indicadas para realizar verificações onde a ordem não é importante. Por exemplo: Verificar quais fornecedores de uma empresa também são clientes.



HashSet<T>

- Principais métodos:
  - **Add:** Adiciona um elemento ao conjunto.
    Sintaxe: <nome_conjunto>.Add(<valor>)
  - **Remove:** Remove um elemento do conjunto por meio de seu valor. **Sintaxe:** <nome_conjunto>.Remove(<valor>)
  - **IsSubsetOf:** Verifica se um conjunto está contido em outro.
    Sintaxe: <nome_conjunto1>.IsSubsetOf(<nome_conjunto2>)
  - **IsSupersetOf:** Verifica se um conjunto contém outro.
    Sintaxe: <nome_conjunto1>.IsSupersetOf(<nome_conjunto2>)

* A classe HashSet<T> pode ser inicializada com valores ao ser declarada. Exemplo:

```
HashSet<string> nomes = new HashSet<string>() {"João","Pedro","Maria","Carlos"};
```

HashSet<T>

- Principais métodos:
  - **IntersectWith:** Retorna os elementos que dois conjuntos têm em comum. O conjunto que chama o método é alterado, passando a conter apenas os elementos em comum.
    Sintaxe: <nome_conjunto1>.IntersectWith(<nome_conjunto2)
  - **UnionWith:** Junta os elementos de dois conjuntos, descartando os elementos repetidos. O conjunto que chama o método é alterado passando a conter os elementos dos dois conjuntos.
    Sintaxe: <nome_conjunto1>.UnionWith(<nome_conjunto2)
  - **ExceptWith:** Retorna os elementos de um conjunto que não estão no outro conjunto. O conjunto que chama o método é alterado passando a conter apenas os elementos que não estão no outro conjunto.
    Sintaxe: <nome_conjunto1>.ExceptWith(<nome_conjunto2)

HashSet<T>

- Método **IntersectWith**

```
João, Pedro, Paulo, Maria, Carlos,
Ana, Paulo, Antonio, Isabel, Maria,
João, Pedro, Paulo, Maria, Carlos, Regina,
Ana, Paulo, Isabel, Maria,
Paulo, Maria,
```

```
class HashSet{
    static void Main(string[] args){
        HashSet<string> clientes = new HashSet<string>() {"João","Pedro","Paulo","Maria","Carlos"};
        HashSet<string> fornecedores = new HashSet<string>() {"Ana","Paulo","Antonio","Isabel","Maria"};

        foreach (string c in clientes) // Imprime a coleção "clientes".
            Console.Write(c + ", "); Console.ReadKey(); Console.WriteLine();
        foreach (string f in fornecedores) // Imprime a coleção "fornecedores".
            Console.Write(f + ", "); Console.ReadKey(); Console.WriteLine();

        // Adiciona o elemento "Regina" na coleção "clientes".
        clientes.Add("Regina");
        foreach (string c in clientes) // Imprime a coleção "clientes".
            Console.Write(c + ", "); Console.ReadKey(); Console.WriteLine();

        // Remove o elemento "Antonio" na coleção "fornecedores".
        fornecedores.Remove("Antonio");
        foreach (string f in fornecedores) // Imprime a coleção "fornecedores".
            Console.Write(f + ", "); Console.ReadKey(); Console.WriteLine();

        clientes.IntersectWith(fornecedores);
        foreach (string c in clientes) // Imprime a coleção "clientes".
            Console.Write(c + ", "); Console.ReadKey(); Console.WriteLine();
    }
}
```

l.bertholdo@ifsp.edu.br

HashSet<T>

- Método **UnionWith**

```
class HashSet2{
    static void Main(string[] args){
        HashSet<string> clientes = new HashSet<string>() {"João","Pedro","Paulo","Maria"};
        HashSet<string> fornecedores = new HashSet<string>() {"Ana","Paulo","Maria","Isabel"};

        foreach (string c in clientes) // Imprime a coleção "clientes".
            Console.Write(c + ", "); Console.ReadKey(); Console.WriteLine();
        foreach (string f in fornecedores) // Imprime a coleção "fornecedores".
            Console.Write(f + ", "); Console.ReadKey(); Console.WriteLine();

        clientes.UnionWith(fornecedores);
        foreach (string c in clientes) // Imprime a coleção "clientes".
            Console.Write(c + ", "); Console.ReadKey(); Console.WriteLine();
    }
}
```

```
João, Pedro, Paulo, Maria,
Ana, Paulo, Maria, Isabel,
João, Pedro, Paulo, Maria, Ana, Isabel,
```

l.bertholdo@ifsp.edu.br

HashSet<T>

- Método **ExceptWith**

```
class HashSet2{
    static void Main(string[] args){
        HashSet<string> clientes = new HashSet<string>() {"João","Pedro","Paulo","Maria"};
        HashSet<string> fornecedores = new HashSet<string>() {"Ana","Paulo","Maria","Isabel"};

        foreach (string c in clientes) // Imprime a coleção "clientes".
            Console.Write(c + ", "); Console.ReadKey(); Console.WriteLine();
        foreach (string f in fornecedores) // Imprime a coleção "fornecedores".
            Console.Write(f + ", "); Console.ReadKey(); Console.WriteLine();

        clientes.ExceptWith(fornecedores);
        foreach (string c in clientes) // Imprime a coleção "clientes".
            Console.Write(c + ", "); Console.ReadKey(); Console.WriteLine();
    }
}
```

```
João, Pedro, Paulo, Maria,
Ana, Paulo, Maria, Isabel,
João, Pedro,
```

l.bertholdo@ifsp.edu.br

HashSet<T>

- Métodos **IsSubsetOf** e **IsSupersetOf**

```
class HashSet3{
    static void Main(string[] args){
        HashSet<string> conjunto1 = new HashSet<string>() {"Paulo","Maria"};
        HashSet<string> conjunto2 = new HashSet<string>() {"João","Paulo","Antonio","Maria"};

        foreach (string c in conjunto1) // Imprime a coleção "clientes".
            Console.Write(c + ", "); Console.ReadKey(); Console.WriteLine();
        foreach (string f in conjunto2) // Imprime a coleção "fornecedores".
            Console.Write(f + ", "); Console.ReadKey(); Console.WriteLine();

        if (conjunto1.IsSubsetOf(conjunto2))
            Console.WriteLine("O conjunto 1 está contido no conjunto 2.");
        else
            Console.WriteLine("O conjunto 1 não está contido no conjunto 2."); Console.ReadKey();

        if (conjunto1.IsSupersetOf(conjunto2))
            Console.WriteLine("O conjunto 1 contém o conjunto 2.");
        else
            Console.WriteLine("O conjunto 1 não contém o conjunto 2."); Console.ReadKey();
    }
}
```

```
Paulo, Maria,
João, Paulo, Antonio, Maria,
O conjunto 1 está contido no conjunto 2.
O conjunto 1 não contém o conjunto 2.
```

l.bertholdo@ifsp.edu.br

Outras Coleções Genéricas

- **LinkedList<T>:** Implementa uma lista duplamente encadeada. Assim, cada elemento da coleção possui um ponteiro para o elemento antecessor e outro para o elemento sucessor. Em comparação com a coleção **List**, possui mais recursos para manipular os elementos da lista.
- **SortedDictionary<Tkey, TValue>** e **SortedList<Tkey, TValue>:** Similares à coleção **Dictionary**, porém ordenam a coleção pelas chaves. Em geral, a coleção **SortedDictionary** é mais rápida para inserir dados e a coleção **SortedList** mais rápida para recuperá-los.
- **SortedSet<T>:** Similar à coleção **HashSet**, porém tem a capacidade de manter os elementos ordenados.

Classes de Coleção Não Genéricas

- Essas coleções são consideradas “não genéricas” pois estão vinculadas ao tipo *Object*.
- Como os elementos são “objetos”, ao armazená-los, essas classes precisam converter o tipo de dado do elemento (int, float, string, etc) para o tipo *Object* e, ao recuperá-los, precisam fazer a operação inversa, tornando-se menos eficientes do que as classes genéricas.
- Estas classes eram usadas quando o C# ainda não suportava as coleções genéricas. Por serem obsoletas, permanecem na linguagem apenas para manter a compatibilidade da plataforma com sistemas legados, não sendo recomendado seu uso para novas aplicações.

```
ArrayList nomes = new ArrayList() { "João", "Maria" };
nomes.Add("Pedro"); // Converte string para Object.
foreach (Object n in nomes)
    Console.WriteLine(n); // Converte Object para string.
```

Classes de Coleção Não Genéricas

- Principais Classes

Coleção Não Genérica	Descrição	Coleção Genérica Correspondente
ArrayList	Coleção sequencial de elementos, que aumenta dinamicamente quando se adiciona novos elementos, não sendo necessário definir sua capacidade na inicialização.	List<T>
Queue	Coleção que usa o conceito de fila *first-in-first-out* (FIFO).	Queue<T>
Stack	Coleção que usa o conceito de pilha *last-in-first-out* (LIFO).	Stack<T>
HashTable	Coleção de pares "chave-valor" que são organizados com base no código *hash* da chave.	Dictionary<TKey, TValue>
SortedList	Coleção de pares "chave-valor", que são ordenados pelas chaves e são acessíveis por chave e por índice.	SortedList<TKey, TValue>

Referências

- Alfredo Lotar; ASP.NET com C# – Curso Prático. Novatec, 2003.
- Cláudio Vieira Oliveira, Ângela Lühmann e Benedito Petroni; Visual Studio C# – Fundamentos, Programação com ASP.NET, Windows Forms e Web Services. Editora Ciência Moderna, 2015.
- John Sharp; Microsoft Visual C# 2013: Passo a Passo. Bookman, 2014.
- http://www.macoratti.net/12/12/c_col1.htm