Coleções .NET

Algoritmos e Estruturas de Dados

Algoritmos e Estruturas de Dados

Rodrigo Richard Gomes

Estruturas de dados básicas

- A plataforma .NET fornece classes de **estruturas de dados** pré-empacotadas
- Essas classes são conhecidas como **classes de coleção**
- Ao utilizar essas classes, nós utilizamos as estruturas de dados sem nos preocuparmos de que maneira elas foram implementadas (*reutilização de código*)
- Necessário usar o namespace System.Collections

Estruturas de dados básicas

- Algumas coleções disponíveis na plataforma .NET
  - ArrayList
  - Queue
  - Stack
  - Hashtable
  - SortedList

Estruturas de dados básicas

- Classe **ArrayList**
  - Frequentemente, precisamos usar vetores sem, entretanto, saber o tamanho ideal
  - A classe *ArrayList* imita um vetor convencional e, adicionalmente, fornece redimensionamento dinâmico
  - Um objeto *ArrayList* pode conter uma quantidade de elementos menor ou igual à sua capacidade

Estruturas de dados básicas

- Classe **ArrayList**
  - A capacidade de um *ArrayList* pode ser manipulada através de sua propriedade *Capacity*
  - Se um *ArrayList* precisa crescer, por padrão ele duplica sua capacidade (*Capacity*) atual
  - *ArrayLists* armazenam referências para objetos, sendo assim, podem conter objetos de qualquer tipo

Estruturas de dados básicas

- Classe **ArrayList**
  - Criação de um ArrayList sem informar a capacidade
    - Ex: `ArrayList AL = new ArrayList();`
  - Criação de um ArrayList informando a capacidade
    - Ex: `ArrayList AL = new ArrayList(10);`

Estruturas de dados básicas

- Classe **ArrayList**
  - A capacidade (Capacity) e a quantidade de elementos (Count) inicial de um *ArrayList* é ZERO
  - Após inserir o primeiro elemento, a capacidade é igual a 4
  - Exercício: faça um pequeno programa que crie um *ArrayList* e imprima sua capacidade e quantidade de elementos. Adicione um elemento e repita a impressão.

Estruturas de dados básicas

- Classe **ArrayList**

```
using System;
using System.Collections;
namespace ExemploArrayList
{
    class Program
    {
        static void Main(string[] args)
        {
            ArrayList AL = new ArrayList();
            Console.Write("AL.Capacity = {0} - AL.Count = {1}\n", AL.Capacity, AL.Count);
            AL.Add(1);
            Console.Write("AL.Capacity = {0} - AL.Count = {1}\n", AL.Capacity, AL.Count);
            Console.ReadKey();
        }
    }
}
```

Método	Descrição
Add	Adiciona um objeto no *ArrayList* e retorna sua posição
BinarySearch	Realiza uma busca binária e retorna a posição do elemento (à partir de 0) ou um número negativo se não for encontrado
Clear	Remove todos os elementos do *ArrayList* sem alterar a capacidade
Contains	Retorna *true* se um objeto especificado estiver no *ArrayList*
Count	Retorna a quantidade de elementos do *ArrayList*
IndexOf	Retorna o índice da primeira ocorrência de um objeto especificado no *ArrayList*
Insert	Insere um objeto na posição especificada. Ocorre uma exceção se a posição não existir.
LastIndexOf	Retorna o índice da última ocorrência de um objeto especificado no *ArrayList*
Remove	Remove a primeira ocorrência do objeto especificado
RemoveAt	Remove um objeto no índice especificado
RemoveRange	Remove uma determinada quantidade de elementos à partir do índice especificado
Reverse	Inverte a ordem dos elementos do *ArrayList*
Sort	Ordena o *ArrayList*
ToArray	Copia os elementos do *ArrayList* para um *Array*
TrimToSize	Altera a capacidade do *ArrayList* para a quantidade de elementos contidos atualmente

Estruturas de dados básicas

Método	Exemplo
Add	AL.Add(15); AL.Add(3.14159); AL.Add(“AED”);
BinarySearch	Posicao = AL.BinarySearch(”AED”);
Clear	AL.Clear()
Contains	if (AL.Contains(15)) Console.Write(“Elemento encontrado”);
Count	Qtde = AL.Count;
IndexOf	Pos15 = AL.IndexOf(15);
Insert	AL.Insert(2, 125); // Adiciona 125 na posição 2
LastIndexOf	Pos15 = AL.LastIndexOf(15);
Remove	AL.Remove(3.14159); // Não ocorre exceção se elemento inexistente
RemoveAt	AL.RemoveAt(1); // Ocorre exceção de posição inexistente
RemoveRange	AL.RemoveRange(0, 2); // Remove 2 elementos à partir da posição 0. Ocorre exceção se não existir a quantidade de elementos desejada

Reverse	AL.Reverse(); // Inverte os elementos de todo o *ArrayList*
	AL.Reverse(3,5); // Inverte os 5 elementos à partir da posição 3

Estruturas de dados básicas

Método	Exemplo
Sort	AL.Sort();
ToArray	Object[] vetor; ... vetor = AL.ToArray();
TrimToSize	AL.TrimToSize();

Estruturas de dados básicas

- Classe **Queue**
  - Implementa a estrutura de dados FILA
  - Filas são estruturas do tipo FIFO (*First-in First-out*)
  - Operações básicas de uma pilha : *enqueue* e *dequeue*
  - *Enqueue* recebe um objeto como argumento e o adiciona no fim da FILA (enfileiramento)
  - *Dequeue* remove e retorna o objeto que está no início da FILA

Estruturas de dados básicas

Método	Descrição
Clear	Remove todos os elementos do *Queue*
Contains	Retorna *true* se um objeto especificado estiver no *Queue*
Count	Retorna a quantidade de elementos no *Queue*
Dequeue	Remove e retorna o objeto posicionado no início do *Queue*
Enqueue	Adiciona um objeto no final do *Queue*
Peek	Retorna o objeto posicionado no início do *Queue* sem removê-lo

Estruturas de dados básicas

- Classe **Stack**
  - Implementa a estrutura de dados PILHA
  - Pilhas são estruturas do tipo LIFO (*Last-in First-out*)
  - Operações básicas de uma pilha : *push* e *pop*
  - *Push* recebe um objeto como argumento e o empilha no topo da pilha
  - *Pop* remove e retorna o objeto que está no topo da pilha

Estruturas de dados básicas

Método	Descrição
Clear	Remove todos os elementos do *Stack*
Contains	Retorna *true* se um objeto especificado estiver no *Stack*
Count	Retorna a quantidade de elementos do *Stack*
Peek	Retorna uma elemento sem removê-lo do topo da PILHA
Pop	Retorna e remove um elemento do topo da PILHA
Push	Adiciona um elemento ao topo da PILHA

Como iterar?

- ArrayList
  - Podem ser utilizados tanto os comandos de repetição for/while/do while quanto o comando foreach
  - Exemplo

```
ArrayList AL = new ArrayList();
…
for(int i = 0; i < AL.Count; i++)
   Console.WriteLine(AL[i]);
foreach(object o in AL)
   Console.WriteLine(o);
foreach(int num in AL)
   Soma = Soma + num;
```

Como iterar?

- ArrayList
  - Podem ser utilizados tanto os comandos de repetição for/while/do while quanto o comando foreach
  - Exemplo

```
ArrayList AL = new ArrayList();
…
for(int i = 0; i < AL.Count; i++)
   Console.WriteLine(AL[i]);
foreach(object o in AL)
   Console.WriteLine(o);
foreach(int num in AL)
   Soma = Soma + num;
```

Como iterar?

- ArrayList
  - Podem ser utilizados tanto os comandos de repetição for/while/do while quanto o comando foreach
  - Exemplo

```
ArrayList AL = new ArrayList();
…
for(int i = 0; i < AL.Count; i++)
   Console.WriteLine(AL[i]);
foreach(object o in AL)
   Console.WriteLine(o);
foreach(int num in AL)
   Soma = Soma + num;
```

Como iterar?

- ArrayList
  - Podem ser utilizados tanto os comandos de repetição for/while/do while quanto o comando foreach
  - Exemplo

```
ArrayList AL = new ArrayList();
…
for(int i = 0; i < AL.Count; i++)
   Console.WriteLine(AL[i]);
foreach(object o in AL)
   Console.WriteLine(o);
foreach(int num in AL)
   Soma = Soma + num;
```

Como iterar?

- ArrayList
  - Podem ser utilizados tanto os comandos de repetição for/while/do while quanto o comando foreach
  - Exemplo

```
ArrayList AL = new ArrayList();
…
for(int i = 0; i < AL.Count; i++)
   Console.WriteLine(AL[i]);
foreach(object o in AL)
   Console.WriteLine(o);
foreach(int num in AL)
   Soma = Soma + num;
```

Como iterar?

- Queue e Stack
  - Essas classes **não podem ser iteradas como um vetor**, assim como pode ser feito com ArrayList's
  - PORTANTO, SÓ É POSSÍVEL ITERAR Queue's e Stack's através do comando foreach
  - Exemplo

```
Queue Q = new Queue();
…
for(int i = 0; i < Q.Count; i++)
   Console.WriteLine(Q[i]);
foreach(object o in Q)
   Console.WriteLine(o);
foreach(int num in Q)
   Soma = Soma + num;
```

Como iterar?

- Queue e Stack
  - Essas classes não podem ser iteradas como um vetor, assim como pode ser feito com ArrayList's
  - PORTANTO, SÓ É POSSÍVEL ITERAR Queue's e Stack's através do comando foreach
  - Exemplo

```
Queue Q = new Queue();
…
for(int i = 0; i < Q.Count; i++)
   Console.WriteLine(Q[i]);
foreach(object o in Q)
   Console.WriteLine(o);
foreach(int num in Q)
   Soma = Soma + num;
```

Como iterar?

- Queue e Stack
  - Essas classes não podem ser iteradas como um vetor, assim como pode ser feito com ArrayList's
  - PORTANTO, SÓ É POSSÍVEL ITERAR Queue's e Stack's através do comando foreach
  - Exemplo

```
Queue Q = new Queue();
…
for(int i = 0; i < Q.Count; i++)
   Console.WriteLine(Q[i]);
foreach(object o in Q)
   Console.WriteLine(o);
foreach(int num in Q)
   Soma = Soma + num;
```

Como iterar?

- Queue e Stack
  - Essas classes não podem ser iteradas como um vetor, assim como pode ser feito com ArrayList's
  - PORTANTO, SÓ É POSSÍVEL ITERAR Queue's e Stack's através do comando foreach
  - Exemplo

```
Queue Q = new Queue();
…
for(int i = 0; i < Q.Count; i++)
   Console.WriteLine(Q[i]);
foreach(object o in Q)
   Console.WriteLine(o);
foreach(int num in Q)
   Soma = Soma + num;
```

Como iterar?

- O mesmo se aplica ao Stack
  - Exemplo

```
Stack S = new Stack();
…
for(int i = 0; i < S.Count; i++)
   Console.WriteLine(S[i]);
foreach(object o in S)
   Console.WriteLine(o);
foreach(int num in S)
   Soma = Soma + num;
```

Estruturas de dados básicas

- Classe **Hashtable**
  - Representa uma estrutura do tipo Dicionário/Mapa composta por pares chave/valor (*key/value*) que são organizados com base no código *hash* da chave
  - Situação: Você precisa pesquisar eficientemente os dados dos 1000 funcionários de uma empresa. Se você quisesse utilizar os 11 dígitos do CPF como índice para armazenar e pesquisar tais dados, precisaria criar um vetor com 999.999.999 posições
  - O problema é que as chaves estarão espalhadas em um intervalo muito grande

Estruturas de dados básicas

- Classe **Hashtable**
  - A solução é converter as chaves (no exemplo anterior, o CPF) em índices exclusivos no vetor, o que é a base da técnica de *hashing*
  - Uma função de *hash* efetua um cálculo para determinar a posição de determinado dado no vetor

Estruturas de dados básicas

Método	Descrição
Add	Adiciona um objeto com chave e valores determinados na tabela *hash*
Clear	Remove todos os elementos da tabela *hash*
Contains	Verifica se a tabela *hash* contém um objeto com determinada chave
ContainsKey	Idem Contains
ContainsValue	Verifica se a tabela *hash* contém um objeto com determinado valor
Count	Retorna a quantidade de elementos armazenados na tabela *hash*
Remove	Remove o elemento especificado da tabela *hash*

Como iterar?

- Hashtable
  - Assim como Queue e Stack, objetos da classe Hashtable não podem ser iterados como um vetor
  - PORTANTO, SÓ É POSSÍVEL ITERAR Hastable através do comando foreach

Como iterar?

- Iterando sobre as chaves

```
foreach (int chave in HT.Keys)
   Console.Write(chave+" ");
```

- Iterando sobre os valores

```
foreach (String valor in HT.Values)
   Console.Write(valor +" ");
```

Como iterar?

- Iterando sobre as chaves e valores

```
foreach (DictionaryEntry DE in HT)
    Console.WriteLine(DE.Key+"\t"+DE.Value);
```

Estruturas de dados básicas

- Classe **SortedList**
  - Representa uma estrutura do tipo Dicionário/Mapa composta por pares chave/valor (*key/value*) ordenados pela chave e acessíveis tanto pela chave quanto pelo índice

Estruturas de dados básicas

Método	Descrição
Add	Adiciona um objeto com chave e valores determinados
Clear	Remove todos os elementos
Contains	Verifica se a tabela contém um objeto com determinada chave
ContainsKey	Idem Contains
ContainsValue	Verifica se a tabela contém um objeto com determinado valor
Count	Retorna a quantidade de elementos armazenados na tabela
Remove	Remove o elemento especificado da tabela
RemoveAt	Remove o elemento armazenado no índice especificado
GetByIndex	Retorna o valor armazenado no índice especificado
GetKey	Retorna a chave armazenada no índice especificado
IndexOfKey	Retorna o índice da chave especificada
IndexOfValue	Retorna o índice do valor especificado
SetByIndex	Trocar o valor armazenado no índice especificado

Como iterar?

- SortedList
  - Pode-se iterar usando tanto o comando for quanto o comando foreach

Como iterar?

```
// Iterando sobre as chaves - usando foreach
Console.WriteLine("\nCHAVES\n======");
foreach (Object chave in SL.Keys)
    Console.Write(chave + " ");

// Iterando sobre os valores - usando foreach
Console.WriteLine("\n\nVALORES\n=======");
foreach (Object valor in SL.Values)
    Console.WriteLine(valor + " ");

// Iterando sobre chaves e valores - usando foreach
Console.WriteLine("\nCHAVES E VALORES\n================");
Console.WriteLine("Chave\tValor");
foreach (DictionaryEntry DE in SL)
    Console.WriteLine(DE.Key + "\t" + DE.Value);
```

Como iterar?

```
 // Iterando sobre as chaves - usando for
Console.WriteLine("\nCHAVES\n======");
for(int i =0; i < SL.Count; i++)
    Console.Write(SL.GetKey(i) + " ");

// Iterando sobre os valores - usando for
Console.WriteLine("\n\nVALORES\n=======");
for (int i = 0; i < SL.Count; i++)
    Console.WriteLine(SL.GetByIndex(i));

// Iterando sobre chaves e valores - usando for
Console.WriteLine("\nCHAVES E VALORES\n================");
Console.WriteLine("Chave\tValor");
for (int i = 0; i < SL.Count; i++)
    Console.WriteLine(SL.GetKey(i)+"\t"+SL.GetByIndex(i));
```

Bibliografia

- SHARP, John – Microsoft Visual C# 2013 passo a passo – Capítulo 18
- DEITEL, H. – C# Como Programar – Capítulo 23
- SCHILDT, Herbert – C# 4.0: The Complete Reference – Capítulo 24
- http://www.dotnetperls.com/
- http://msdn.microsoft.com/en-us/library/k166wx47 (MSDN)