Collections

Esta parte do documento C# Essencial, dá início ao estudo das colecções na linguagem C# nomeadamente o estudo das classes *ArrayList*, *Stack, Queue, HashTable, SortedList* definidas no namespace *System.Collections* (*using System.Collections*;).

Sumário:

ArrayList

A classe *ArrayList* é semelhante aos *arrays*. No entanto o seu tamanho é alterado dinamicamente à medida que os elementos são alterados. A instrução seguinte mostra como criar um objecto desta classe:

```
ArrayList list = new ArrayList();
```

O método *Add* permite adicionar elementos nos objectos da classe *ArrayList*:

```
list.Add(45);
list.Add(87);
list.Add(12);
```

O acesso aos dados do objecto pode ser efectuado como se de um array se tratasse (a instrução *Count* permite obter o número total de elementos presentes no objecto). O código seguinte implementa a listagem de todos os elementos de um ArrayList.

```
static void Main()
{
    ArrayList list = new ArrayList();
    list.Add(45);
    list.Add(87);
    list.Add(12);
    for (int i = 0; i < list.Count; i++)
    {
        Console.WriteLine(list[i]);
    }
}
```

Métodos/propriedades definidos na classe *ArrayList* :

- **Capacity** – devolve ou altera o número de elementos que a ArrayList pode conter.
- **Count** – número de elementos na ArrayList.
- **Add(object obj)** – adiciona o elemento obj na ArrayList.
- **Remove(object obj)** – remove o elemento obj da ArrayList.
- **RemoveAt(int i)** – remove um elemento da ArrayList na posição i
- **Insert(int i, object obj)** - insere o elemento obj na ArrayList na posição i.
- **Clear()** - Remove todos os elementos da ArrayList
- **Contains(object obj)** – Devolve um valor lógica consoante a ArrayList contêm o elemento obj ou não.
- **IndexOf(object obj)** – Devolve o índice da primeira ocorrência de obj na ArrayList. Se não existir devolve -1

Stack

A classe *Stack* implementa o princípio *LIFO (Last In First Out).* A instrução seguinte mostra como criar um objecto desta classe:

```
Stack st = new Stack();
```

Métodos definidos na classe *Stack*:

- **Push(object obj)** – adiciona o elemento obj no topo da pilha.
- **Pop()** – remove o elemento do topo da pilha.
- **Peek()** – devolve o elemento do topo da pilha.

```
static void Main()
{
    Stack stack = new Stack();
    stack.Push(2);
    stack.Push(4);
    stack.Push(6);
    Console.WriteLine("Número de elementos antes de Peek(): {0}", stack.Count);
    Console.WriteLine("Elemento Topo da Stack: {0}", stack.Peek());
    Console.WriteLine("Número de elementos após Peek(): {0}", stack.Count);

}
```

Queue

A classe *Queue* implementa o princípio *FIFO (First In First Out)*. A instrução seguinte mostra como criar um objecto desta classe:

```
Queue q = new Queue();
```

A classe Queue implementa os métodos *Enqueue* e *Dequeue* para, respectivamente, adicionar um novo elemento no fim da fila e retirar o primeiro elemento da fila.

```
static void Main()
{
    Queue queue = new Queue();
    queue.Enqueue(2);
    queue.Enqueue(4);
    queue.Enqueue(6);
    while (queue.Count != 0)
    {
        Console.WriteLine(queue.Dequeue());
    }
}
```

HashTable

A classe *HashTable* permite manipular pares chave/valor. Cada valor de uma *HashTable* é identificado pela sua chave. Numa *HashTable* todas as chaves são únicas. O conceito é semelhante a um dicionário onde cada palavra (chave) está associada a sua definição (valor). A instrução seguinte mostra como criar um objecto desta classe:

```
HashTable ht = new HashTable();
```

O método *Add(object chave, object valor)* permite adicionar um novo elemento (chave/valor) numa *HashTable*:

```
ht.Add("st01", "Faraz");
ht.Add("sci01", "Newton");
ht.Add("sci02", "Einstein");
```

O método Count devolve o número de elementos (chave/valor) presentes na HashTable. O exemplo seguinte exemplifica o acesso ao valor de um par a partir da sua chave (*ht["st01"]*):

```
Console.WriteLine("Tamanho da Hashtable:{0}", ht.Count);
Console.WriteLine("Elemento com a chave: st01 é:  {0}", ht["st01"]);
```

A remoção de um par a partir da sua chave é realizada através do método *Remove(Object obj)*. Vejamos o seguinte exemplo:

```
static void Main()
{
    Hashtable ht = new Hashtable(20);
    ht.Add("st01", "Faraz");
    ht.Add("sci01", "Newton");
    ht.Add("sci02", "Einstein");
    Console.WriteLine("Tamanho da Hashtable é: {0}", ht.Count);
    Console.WriteLine("Remover o elemento com a chave st01");
    ht.Remove("st01");
    Console.WriteLine("Tamanho da Hashtable após remoção: {0}", ht.Count);
}
```

As instruções *Keys* e *Values* permitem obter, respectivamente, o conjunto das chaves e o conjunto dos valores de uma HashTable:

```
static void Main()
{
    Hashtable ht = new Hashtable(20);
    ht.Add("st01", "Faraz");
    ht.Add("sci01", "Newton");
    ht.Add("sci02", "Einstein");
    Console.WriteLine("Mostrando chaves...");
    foreach (string key in ht.Keys)
    {
        Console.WriteLine(key);
    }
    Console.WriteLine("\nMostrando valores...");
    foreach (string Value in ht.Values)
    {
        Console.WriteLine(Value);
    }
}
```

O método *ContainsKey(Object obj)* permite determinar se uma chave pertence ou não à *HashTable*. Por sua vez o método *ContainsValue(Object obj)* permite determinar se uma chave pertence ou não à *HashTable*.

```
static void Main()
{
    Hashtable ht = new Hashtable(20);
    ht.Add("st01", "Faraz");
    ht.Add("sci01", "Newton");
    ht.Add("sci02", "Einstein");
    Console.WriteLine(ht.ContainsKey("sci01"));
    Console.WriteLine(ht.ContainsKey("st00"));
    Console.WriteLine(ht.ContainsValue("Einstein"));
}
```

Neste caso o resultado apresentado seria:

True

False

True.

SortedList

A classe *SortedList* permite manipular pares chave/valor, ordenando a informação pela chave. Os dados podem ser acedidos a partir das chaves ou a partir dos índices dos pares. A instrução seguinte mostra como criar um objecto desta classe:

```
SortedList sl = new SortedList();
```

Propriedades e métodos definidos na classe:

- **Count** – número de elementos contidos na SortedList.
- **Keys** – devolve o conjunto das chaves da SortedList.
- **Values** – devolve o conjunto dos valores da SortedList.
- **Add(object key, object value)** – adiciona um novo par (chave, valor) na SortedList.
- **GetKey(int index)** – devolve a chave presente na pc
- **GetByIndex(int index)** – devolve o valor presente na posição
- **IndexOfKey(object key)** – devolve o índice de uma chave.
- **IndexOfValue(object value)** – devolve o índice de um valor.
- **Remove(object key)** – remove um par (chave/valor) a partir da sua chave.
- **RemoveAt(int)** – remove um par a partir do seu índice.
- **Clear()** - remove todos os pares da SortedList.
- **ContainsKey(object key)** – Devolve um valor lógico consoante a existência ou não da chave *key* no SortedList.

- **ContainsValue(object value)** - Devolve um valor lógico consoante a existência ou não do valor *value* no SortedList

Exercício 1:

Resolva este exercício fazendo uso da classe *SortedList*

1. Pretende-se implementar uma agenda de contactos telefónicos.

   Para representação de um contacto define a classe *Contacto* com duas variáveis de instância: *nome* e *telemóvel*;

   Na classe *Contacto* define ainda os métodos/construtores seguintes:

   ```
   Contacto()
   Contacto(string nome, int telemóvel)
   public string getNome(); //obter nome
   public string getTelemóvel(); //obter telemóvel
   public void setNome(string nome); //alterar nome
   public void setTelefone(int telemóvel); //alterar telemóvel
   public bool igual(Contacto c); //determinar igualdade entre dois contactos
   public string toString(); //extrair dados de um contacto sob a forma de uma string
   ```

2. Define agora uma nova classe *Agenda* contendo uma identificação e uma lista de contactos

   Na agenda não podem existir contactos repetidos.

   Na classe *Agenda* define ainda os métodos/construtores seguintes:

   ```
   Agenda(string identificação);
   int total(); //número total de contactos
   void addContacto(Contacto c); //adicionar novo contacto
   void addContacto(string nome, int telemóvel); //adicionar novo contacto
   void remove(string nome); //remover contacto
   bool existe(string nome); //determinar existência de contacto
   bool existe(Contacto c); //determinar existência de contacto
   int obterContacto(string nome); //obter telemóvel
   string obterNomes(int telemóvel); //obter nome
   bool igual(Agenda a) ; //igualdade entre duas agendas
   Agenda ordena(); //ordenar agenda
   ```

3. Define uma classe de teste contendo duas agendas.

4. Crie um menu interactivo para manipular as agendas.

Exercício 2:

Define um sistema para gestão de um stand automóvel. Para cada automóvel o sistema deverá armazenar os dados seguintes:

- matrícula
- marca
- modelo
- cilindrada
- quilómetros
- preço
- estado (a venda ou vendido)

Cada cliente do stand deverá ficar armazenado no sistema com a informação seguinte:

- número de bilhete de identidade
- nome
- morada
- lista das matrículas dos veículos comprados no stand

Desenvolva uma aplicação com as funcionalidades seguintes:

- inserção de um novo automóvel no stand
- registo da venda de um automóvel

A informação dos automóveis deverá ser armazenada numa *SortedList*. Os dados dos clientes devem ser inseridos numa *ArrayList*, guardando as matrículas dos veículos comprados numa *Queue*.

Referências

http://www.softsteel.co.uk/tutorials/cSharp/

http://www.csharp-station.com/Tutorial.aspx

http://csharpcomputing.com/Tutorials/TOC.htm

http://msdn2.microsoft.com/en-us/library/9b9dty7d(VS.80).aspx

http://msdn2.microsoft.com/en-us/library/2s05feca(VS.80).aspx

C# School – Programmers Heaven (cf. Site da disciplina)