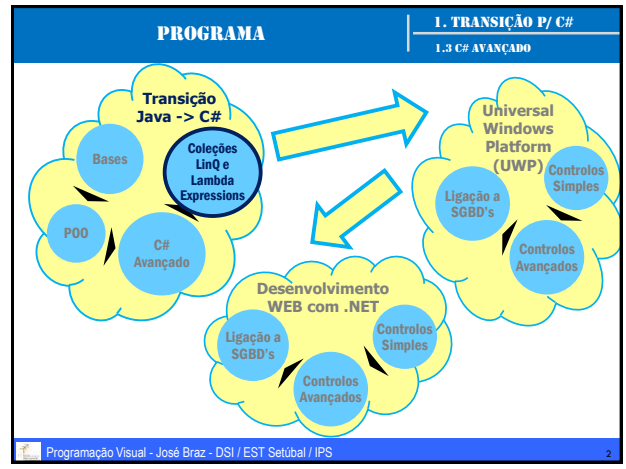
Programação Visual

Coleções 2 de 2 – HashSet – Dictionary – LINQ – Lambda Expressions

1



2

C# – Coleções & LINQ

- Classes de Colecção
  - System.Collections.Generic
    - List<T>
    - HashSet<T>
    - Dictionary<TKey,TValue>
  - System.Collections.Classes
    - ArrayList
  - System.Collections.Concurrent
    - Não veremos em POO
- LINQ
  - Conceitos Básicos
  - Execução Diferida (é standard)
  - Execução Imediata (forçada)
  - Operadores Interrogativos LINQ (query operators)
  - Projecção de dados com tipos anónimos
  - Interrogações LINQ com Expressões Lambda

3

C# – Coleções

❑ **Usamos coleções para armazenar grupos de elementos**

- Os objetos do tipo coleção devem ter como identificador um nome plural

```
Alunos / Turma
```

- Uma coleção é uma classe pelo que temos de declarar e criar um instância dessa classe …

```
var alunos = new List<Aluno>();
```

- antes de lhe adicionar elementos

```
alunos.Add( new Aluno ()
   {  Nome = "Jose Antunes",
      Numero = 1234};
```

- Interressam-nos 3 tipos de coleções:
  - **Listas** : coleções **ordenadas pela ordem de inserção e com possível repetição** de elementos. (List<T> , Stack<T>, SortedList<T> ou ArrayList)
  - **Conjuntos** : coleções **não ordenadas e sem elementos repetidos**. (HashSet<T>, SortedSet<T>)

```
var numerosDeAluno =
      new HashSet<int>();
```

  - **Tabelas** : coleções de **pares (chave, valor)** em que não existem chaves repetidas (Dictionary<TKey, TValue>, SortedDictionary<TKey,TValue)

```
var alunosDaEST =
   new Dictionary <int , Aluno>();
```


4

C# – Coleções – List<T>

```
static void Main(string[] args){
   Console.WriteLine("XXXXXXXXXXXXX Slide 5 -List<T>");

   // Uma List<T> é uma coleção parametrizada (genérica)…
   List<String> nomes = new List<String>();

   Console.WriteLine("*** A Lista de nomes: ");
   // 1. Ordenada : Tem como ordem a ordem de adição de elementos
   nomes.Add("Alberto");
   nomes.Add("Bernardo");
   nomes.Add("Dulce");
   // 2. Pode ter elementos repetidos
   nomes.Add("Dulce"); // repetido ;-)
   foreach (String s in nomes)
      Console.WriteLine(s);
   // 3. É indexada, isto é, podemos aceder a cada
   //    elemento da Lista através do indice da posição
   //    desse elemento

   Console.WriteLine( "*** O elemento na posição 2: " + nomes[2]);
}
```


5

C# – Coleções – List<T>

Propriedades

- Capacity
- Count
- Item[ ]

Métodos

Add
AddRange
AsReadOnly
BinarySearch
Clear
Contains
ConvertAll
CopyTo
Exists

- Find
- FindAll
- FindIndex
- FindLast
- FindLastIndex
- ForEach
- GetEnumerator
- GetRange

IndexOf
Insert
InsertRange
LastIndexOf
Remove
RemoveAll
RemoveAt
RemoveRange
Reverse
Sort
ToArray
TrimExcess
TrueForAll

6

C# – Coleções – Alunos é uma List<Aluno>

```
public class Alunos : List<Aluno> {
   // Como Alunos é uma List<Aluno>
   // 1. Tem já todos os métodos de List<T>
   // 2. Só temos de codificar o ToString
   override public String ToString(){
        String str = "";
        foreach (Aluno a in this){
           str += a + "\n";
        } return str;
   }
}
```

Alunos é uma (herda de) Lista de Aluno. E assim sendo já tem todos os métodos de uma List

```
static void Main(string[] args){
   Console.WriteLine("\nXXXXXX Slide 7 - Alunos : List<Aluno>");

   Alunos alunos = new Alunos();
   Console.WriteLine("A Lista Alunos : List<Aluno>");

   alunos.Add(new Aluno() { Nome = "Armando",  Numero = "190210010" });
   alunos.Add(new Aluno() { Nome = "Armando",  Numero = "190210010" });
   alunos.Add(new Aluno() { Nome = "Beatriz",  Numero = "190210011" });
   alunos.Add(new Aluno() { Nome = "Carlos",   Numero = "190210012" });
   Console.WriteLine("A Lista Alunos com quatro alunos");
   Console.WriteLine(alunos.ToString());
}
```


7

C# – Coleções – Turma tem uma List<Aluno>

```
public class Turma{
    public String Curso { get; set; }
    public int Ano { get; set; }
    public String Nome { get; set; }
    // Turma tem um atributo (Propriedade) do tipo List<Aluno>
    public List<Aluno> Alunos { get; set; }
    public Turma(String curso, int ano, String nome){
        // Como não recebe nenhum parametro para uma List<Aluno>
        // o construtor deve criar uma instancia de List<Aluno>
        Alunos = new List<Aluno>();
    }
}
```

Como a propriedade Alunos da classe Turma é pública podemos ler e escrever nela diretamente

```
static void Main(string[] args){
   Console.WriteLine("\nXXXXXX Slide 8 - Turma tem uma List<Aluno>");
   Turma turma = new Turma("TPSI", 2, "EST Barreiro");

   turma.Alunos.Add(new Aluno() { Nome = "Armando",  Numero = "190210010" });
   turma.Alunos.Add(new Aluno() { Nome = "Armando",  Numero = "190210010" });
   turma.Alunos.Add(new Aluno() { Nome = "Beatriz",  Numero = "190210011" });
   turma.Alunos.Add(new Aluno() { Nome = "Carlos",   Numero = "190210012" });
   Console.WriteLine("A Turma com quatro alunos");
   Console.WriteLine(turma.ToString());
}
```


8

C# – Coleções – HashSet<String>

```
static void Main(string[] args){
   Console.WriteLine("\nXXXXXXXXXXXXXX Slide 9 - HashSet<String>");
   // Um conjunto (Set) é uma coleção que
   // 1. não tem elementos repetidos
   // 2. Os elementos podem não estar pela ordem de inserção
   // 3. não é indexada (não há indice para aceder aos elementos)
   HashSet<String> numerosDeAluno = new HashSet<String>();
   numerosDeAluno.Add("190210001");
   numerosDeAluno.Add("190210002");
   numerosDeAluno.Add("190210003");
   numerosDeAluno.Add("190210004");
   numerosDeAluno.Add("190210004"); // Não adiciona outro 0004
   numerosDeAluno.Add("190210003"); // Não adiciona outro 0003
   // A classe HashSet<String> deteta strings iguais porque
   // a classe String já define os seus próprios Equals e HashCode
   Console.WriteLine("O HashSet<String> sem 04 e 03 repetidos");
   foreach(String s in numerosDeAluno)
       Console.WriteLine(s);
}
```


9

C# – Coleções – HashSet<T>

Propriedades

Comparer
Count

Métodos

Add
Clear
Contains
CopyTo
CreateSetComparer
EnsureCapacity
ExceptWith
GetEnumerator
GetObjectData
IntersectWith
IsProperSubsetOf
IsProperSupersetOf
IsSubsetOf
IsSupersetOf
OnDeserialization
Overlaps
Remove
RemoveWhere
SetEquals
SymmetricExceptWith
TrimExcess
TryGetValue
UnionWith

10

C# – Coleções – SetAlunos é um HashSet<Aluno>

```
public class SetAlunos : HashSet<Aluno> {
   // Como SetAlunos é um HashSet<Aluno> tem já todos os métodos de HashSet<T>
   // 1. Mas Aluno herda os Equals e GetHashCode de Object para os quais dois
   //    objetos do tipo Aluno são iguais se referenciarem o mesmo endereço
   // 2. Por isso teremos de redefinir os Equal e GetHashcode
   //    da classe Aluno por forma a que dois alunos sejam iguais
   //    de acordo com os nossos critérios: dois alunos serão
   //    iguais se tiverem o mesmo numero de aluno (ver classe Aluno)
   // restante código omitido }
```

```
public class Aluno {

  // restante código omitidos

  public override int GetHashCode()
  {
     return Numero.GetHashCode();
  }
```

```
  public override bool Equals(object obj) {
    // Is null
    if (Object.ReferenceEquals(null, obj)){
       return false;
    } // Is the same object
    if (Object.ReferenceEquals(this, obj)){
       return true;
    } // Is not the same type
    if (obj.GetType() != this.GetType()){
       return false;
    }
    Aluno a = obj as Aluno;
    return String.Equals(Numero, a.Numero);
  }
}
```


11

C# – Coleções – HashSet<T>

```
public class NumerosDeAlunos : HashSet<String> {

    public NumerosDeAlunos (Unidade unidade,
                            int ano,
                            int qtdAlunos){

        for (int i = 0; i < qtdAlunos; i++)
            this.Add( ano.ToString().Substring(2) +
            "0" +
            (int)unidade +
            i.ToString("0000"));
    }

// restante código omitido
}
```

NumerosDeAlunos é um (herda de) HashSet de Strings

O construtor com 3 parâmetros adiciona qtdAlunos strings no formato AA0UUU#### a si próprio (this)

12

C# – Coleções – HashSet<T>

```
public class NumerosDeAlunos : HashSet<String>
{

   // restante código omitido
   override public String ToString(){
      String str = "";
      foreach (String s in this){
               str += s + "\n";
      }
      return str;
    }

   // não precisamos de recodificar o Equals e o GetHashSet
   // porque a classe String já implementa os seus overrides

}
```

O ToString retorna uma String com todos os elementos do conjunto (this)

13

C# – Iterar um HashSet<T>

```
static void Main(string[] args)
{
   NumerosDeAlunos numeros =
                new NumerosDeAlunos(Unidade.EST_SETUBAL, 2019, 3);

   NumerosDeAlunos.Enumerator it = numeros.GetEnumerator();

   String val = "";

   do {
         val = it.Current;

         Console.WriteLine(val);
   } while (!val.Equals(190210001) && it.MoveNext());

   Console.WriteLine(val);
}
```

Enumerator é um iterador para uma coleção

GetEnumerator cria e retorna um Enumerator sobre a coleção numeros

it.Current retorna o valor atualmente apontado pelo Enumerator

it.MoveNext tenta mover para o elemento seguinte. Se existir um seguinte move e retorna true caso contrário retorna false

https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.hashset-1?view=netframework-4.8

14

C# – Coleções – Dictionary<TKey, TValue>

```
namespace TP04_Collections
{
   enum Lugares { FAL1, FAL2, FAL3, FAL4,
         FBL1, FBL2, FBL3, FBL4, FBL5, FBL6
   }

   class Sala : Dictionary<Lugares, Aluno>
   {

      public override string ToString()
      {
         String str = "";
         foreach (KeyValuePair<Lugares, Aluno> a in this)
            str += a.Key.ToString() + "- "
                   a.Value.ToString() + "\n";
         return str;
      }
   }
}
```

Sala é um (herda de) Dictionary em que as chaves são Lugares e os valores Alunos

Leia-se: para cada par (chave, valor) a

https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2?view=netframework-4.8

15

C# – Coleções – Dictionary<TKey, TValue>

```
namespace TP04_Collections
{
    enum Lugares { F1LA, F1LB, F1LC, F1LD,
        F2LA, F2LB, F2LC, F2LD, F2LE, F2LF
    }

    class Sala : Dictionary<Lugares, Aluno>
    {

     public override string ToString()
        {
            String str = "";
            foreach (var a in this)
                str += a.Key.ToString() + "- " +
                       a.Value.ToString() + "\n";
            return str;
        }
    }
}
```

Podemos simplificar e usar o "Tipo Implicito" var

16

C# – Coleções – Dictionary<Key, Value>

Propriedades

Comparer
Count
Item[ ]
Keys
Values

Métodos

Add
Clear
ContainsKey
ContainsValue
EnsureCapacity
GetEnumerator
GetObjectData
OnDeserialization
Remove
TrimExcess
TryAdd
TryGetValue

Dictionary<TKey,TValue>.KeyCollection
Propriedade:
Count
Métodos
CopyTo
GetEnumerator

Dictionary<TKey,TValue>.ValueCollection
Propriedade:
Count
Métodos
CopyTo
GetEnumerator

17

C# – System.Collections.Generic

- Quando todos os elementos de uma coleção são de um mesmo tipo podemos (e devemos) usar uma coleção genérica
  - Listas
    - **List<T>**
    - **SortedList<T>**
    - **Stack<T>**
    - **Queue<T>**
  - Conjuntos :
    - **HashSet<T>,**
    - **SortedSet<T>**
  - Tabelas :
    - **Dictionary<TKey, TValue>**
    - **SortedDictionary<TKey,TValue)**
- Mais em:
  - https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic?view=netframework-4.7.1

```
var pontos = new List<Ponto>();
```

```
var numeroDoPonto =
      new HashSet<int>();
```

```
var pontosDaFigura =
   new Dictionary <int , Ponto>();
```


18

C# – System.Collections

- As classes deste namespace não armazenam objetos estritamente tipificados (de um determinado tipo) mas antes como um objeto do tipo Object.
- Estas classes só deverão ser usadas quando de todo não for possível usar a sua correspondente genérica.
  - Listas
    - **ArrayList**
    - **Queue**
    - **Stack**
  - Tabelas :
    - **Hashtable**

```
var coisas = new ArrayList ();
coisas.Add(new Ponto(1,2));
coisas.Add(3);
coisas.Add("Ponto");

foreach (Object o in coisas)
        Console.WriteLine(o);

SAIDA:
(1,2)
3
Ponto
```

- Mais em:
  - https://docs.microsoft.com/en-us/dotnet/api/system.collections?view=netframework-4.7.1

19

C# – System.Collections.Concurrent

- A partir do .NET Framework 4.0 as classes do namespace System.Collections.Concurrent oferecem operações "thread-safe" para aceder a elementos de uma coleção concorrencialmente a partir de diferentes threads.
- A partir do .NET 4.0 devemos usar as classes deste namespace em lugar das correspondentes classes dos .Generic e .Collections sempre que se pretenda aceder a uma coleção a partir múltiplas threads concorrentes.
- Mais em:
  - https://docs.microsoft.com/en-us/dotnet/api/system.collections.concurrent?view=netframework-4.7.1
- Thread:
  - A mais pequena sequência de instruções que pode ser gerida por um scheduler (tipicamente um SO). Normalmente uma thread é parte de um processo e pode partilhar com outras threads recursos como a memória, código executável e valores de variáveis enquanto os processos normalmente não o fazem.

20

Conceitos Básicos de LINQ

- O **LINQ** simplifica e uniformiza a forma de obter **informação** a partir **de um conjunto de dados**.
- O conjunto de dados pode ser qualquer coleção como por exemplo **um *array, uma lista* ou uma base de dados.**
- A linguagem LINQ está integrada no C# com uma sintaxe semelhante à da linguagem SQL (mas 'não é SQL').
- **A sintaxe mais simples** de uma instrução LINQ é:

```
from fonteDeDados in coleção
     select elementoASeleccionar
```


21

Conceitos Básicos de LINQ

```
//Dada uma coleção:
int[] nums = new int[] { 1, 2, 4, 7, 12, 14, 15, 16, 18, 21 };
```

```
//Programação em C#
List<int> res =
       new List<int>();

for (int i = 0; i < nums.Length; i++)
  if (nums[i] >= 5 && nums[i] <= 15)
    res.Add(nums[i]);

foreach( int x in res)
  Console.WriteLine("> " + x);
```

```
//Programação em C# com LINQ

var res =  from i
           in nums
           where i >= 5 &&
                 i <= 15
           select i;

foreach (int x in res)
  Console.WriteLine("> " + x);
```


22

LINQ – execução diferida

- A query LINQ é executada no momento em que é invocada e assim o resultado traz sempre os valores atualizados.

```
List<Aluno> als = new List<Aluno> {
    new Aluno() {Nome = "Jose Antunes", Numero = 2345},
    new Aluno() {Nome = "Ana Costa", Numero = 4567},
    new Aluno() {Nome = "Manuel Jose", Numero = 6789},
};
Console.WriteLine("\n\nALUNOS com o nome 'Jose':");
var res = from a in als where a.Nome.Contains("Jose") select a.Nome;

foreach (var a in res)
 Console.WriteLine("> " + a);

Console.WriteLine("\n Jose Pinto adicionado");
als.Add(new Aluno() { Nome = "Jose Pinto", Numero = 9012 });
Console.WriteLine("\n ALUNOS com o nome 'Jose': \n");
foreach (var a in res)
  Console.WriteLine("> " + a);
```

A listagem feita aqui já mostra o aluno "Jose Pinto" introduzido imediatamente antes da invocação da query res.

23

LINQ – execução imediata

- É possível executar imediatamente uma expressão LINQ chamando um dos métodos definidos no tipo **Enumerable** para o resultado da query como por exemplo:
  - **ToArray<T>()**, **ToList<T>()**, **ToDictionary<TSource,TKey>()**).
  - Neste caso obtemos uma imagem dos dados no momento em que a query é definida (*snapshot*).

```
// expressão LINQ com execução imediata
List<String> joses =
      (from a
      in als
      where a.Nome.Contains("Jose")
      select a.Nome).ToList();

Console.WriteLine(
      "\nALUNOS com o nome 'Jose' (2):");

foreach (String n in joses)
       Console.WriteLine("> " + n);
```

Chamou-se o método **ToList()** que faz a **execução imediata** da *query* e converte o resultado para uma lista de *strings*.
Nota: na realidade estamos a invocar o método **ToList<String>()** onde o tipo parametrizador foi omitido porque o compilador o consegue determinar.

24

LINQ - Operadores

- O **LINQ** utiliza na sua sintaxe **operadores** como por exemplo:
  - **from** especifica uma fonte de dados e **in** dá-lhe o contexto.
  - **select** define uma sequência de elementos a partir dos dados,
  - **where** filtra os dados com uma expressão booleana,
  - **orderby** ordenação os dados por um campo da fonte de dados
  - **ascending** e **descending** que permitem (em conjunto com o **orderby)** ordenar os resultados de forma ascendente ou descendente.
- Para além destes operadores existem vários outro tais como **join**, **on**, **equals**, **into**, **group**, **by**, etc.
- Aconselha-se a consulta:
  - https://msdn.microsoft.com/en-us/library/bb310804.aspx
  - https://msdn.microsoft.com/en-us/library/bb397927.aspx
  - https://msdn.microsoft.com/en-us/library/bb397676.aspx

25

LINQ - Operadores

- Para além dos operadores integrados na linguagem existem operadores sobre a forma de métodos de extensão genéricos que se aplicam diretamente aos resultados da *query* como, por exemplo:
  - **ToList<T>()** converte o resultado numa lista de elementos,
  - **ToArray<T>()** converte o resultado num *array* de elementos,
  - **Reverse<T>()** inverte a ordem dos elementos, etc.
- Além dos mencionados encontram-se alguns que fazem operações sobre conjuntos de elementos
  - **Distinct<T>()**
  - **Union<T>()**
  - **Intersect<T>()**
- ou outros que agregam os resultados
  - **Count<T>()**,
  - **Sum<T>()**, **Min<T>()**, **Max<T>()**, etc.).
- Recomenda-se a análise dos vários operadores disponíveis na documentação da linguagem C#.

26

LINQ – Projecção de Dados

- É possível obter como resultado de uma *query* LINQ um conjunto de dados de um tipo anónimo, subconjunto da coleção inquirida, que podem ser úteis quando se pretende analisar apenas uma parte da coleção.

```
// Projecção de dados com variáveis anónimas
List < Disciplina > disc = new List<Disciplina>();

// Adição de várias disciplinas e alunos...
// Código omitido …

var res3 = from d
           in disc
           where d.Nome.Contains("Programacao")
           select new { Nome = d.Nome, NumeroAlunos = d.TotalAlunos
};

Console.WriteLine("\nDISCIPLINAS: \n");
foreach (var r in res3)
  Console.WriteLine("> " + r.Nome + " com " + r.NumeroAlunos);
```

Foi criado um tipo anónimo com os campos **Nome** e **NumeroAlunos** que contém apenas o nome e o total de alunos na disciplina.

27

LINQ – com métodos de extensão

- É possível não utilizar a sintaxe integrada do LINQ definida para a linguagem C#. Neste caso aplicam-se directamente os métodos de extensão sobre as fontes de dados.
- Muitos dos métodos de extensão têm como argumento referências de métodos que são habitualmente chamados para cada elemento do conjunto de dados.
- Nos exemplos mostra-se a sintaxe LINQ com operadores e a mesma operação usando os métodos de extensão. Usam-se expressões lambda para passar os métodos como argumentos

```
var res4 = from a
  in als
  where a.Nome.Contains("Jose")
  select a.Nome;

var res4 =
   als.Where
   (a => a.Nome.Contains("Jose")).Select
   (a => a.Nome);

var res5 = from a
   in als
   orderby a.Nome select a;

var res5 =
   als.OrderBy
   (a => a.Nome).Select (a => a);

var res6 = from a
   in als
   where a.Numero>5000
   select a.Nome;

var res6 =
   als.Where
   (a => a.Numero>5000).Select
   (a => a.Nome);
```


28

C# - Aula

- 1. Abra o Visual Studio e abra estes slides
- 2. Abra o ficheiro TP06_LINQ.txt
- 3. Crie um projeto denominado TP06_LINQ no VS
- 4. i= 4;
- 4.1 Enquanto i<39
- 5. i++;
  - 5.1. Leia o slide [i] !
  - 5.2. Copie o código do ficheiro TP06_LINQ.txt correspondente ao slide [i] para o método main
  - 5.3. Copie as classes necessárias para dentro do seu projeto (e apenas as necessárias para correr o código do slide [i]) Nas classes importadas comente o código desnecessário.
  - 5.4. Leia os comentários e compreenda o que se pretende ilustrar
  - 5.5. Corra o código e tente interpretar o que se está a passar
  - 5.6. Se tiver dúvidas … p e r g u n t e … please!!!
- 6 . Volte ao ponto 4.1 …

29