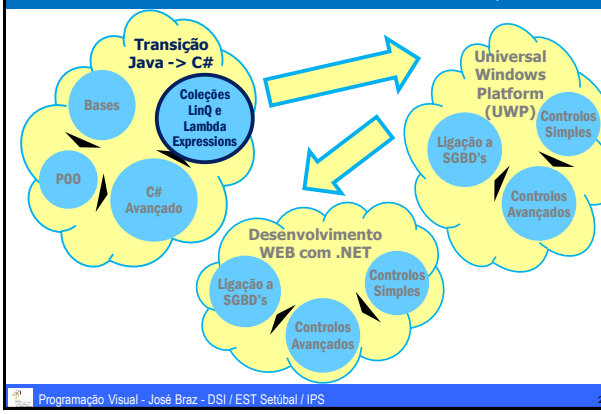
Programação Visual

Coleções – List<T> - HashSet<T> – Dictionary<Tk, Tv>

1



2

C# – Coleções

- Classes de Colecção
  - System.Collections.Generic
    - Listas
      - List < T >
    - Conjuntos
      - HashSet < T >
    - Tabelas
      - Dictionary<TKey,TValue>
        - são os MAP do java
  - System.Collections.Classes
    - ArrayList
  - System.Collections.Concurrent
    - Não veremos em PV

3

C# – Coleções

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

- Os objetos do tipo coleção devem ter como identificador um nome plural (BBP)
- Uma coleção é uma classe pelo que temos de declarar e criar uma instância dessa classe …
- antes de lhe adicionar elementos
- 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>)
  - **Conjuntos** : coleções **não ordenadas e sem elementos repetidos**. (HashSet<T>, SortedSet<T>)
  - **Tabelas** : coleções de **pares (chave, valor)** em que não existem chaves repetidas (Dictionary<TKey, TValue>, SortedDictionary<TKey,TValue)

```
Alunos / Turma

var alunos = new List<Aluno>();

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

var numerosDeAluno =
      new HashSet<int>();

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


4

C# – Coleções

❑ **As classes de coleção (coleções) que vamos abordar são Tipos Genéricos**

- Um tipo de dados genérico é um tipo de dados que pode (tem de) ser parametrizado.

Tipos Genéricos < T >

Tipos parametrizados

```
List< T >                          List< Aluno > alunos;

HashSet < T >                      HashSet < Mesa > mesas;

Dictionary < Tkey, Tvalue >        Dictionary < Aluno, Nome > nomes
```


5

C# – Coleções – List<T>

```
static void Main(string[] args){
   Console.WriteLine("XXXXXXXXXXXXX Slide 6 -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]);
}
```


6

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

7

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());
}
```


8

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 t = new Turma();

   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());
}
```


9

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 GetHashCode
   Console.WriteLine("O HashSet<String> sem 04 e 03 repetidos");
   foreach(String s in numerosDeAluno)
       Console.WriteLine(s);
}
```


10

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

11

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 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 número de aluno (ver classe Aluno)
   // restante código omitido }
```

```
public class Aluno {

  // restante código omitido

  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);
  }
}
```


12

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)

13

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 GetHashCode
   // porque a classe String já implementa os seus override

}
```

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

14

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

15

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

16

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

17

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

18

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>();
```


19

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

```
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
```


20

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.

21