
Programação Web

Aulas Teóricas – Capítulo 1 – 1.3

1º Semestre - 2023/2024

Departamento de Engenharia Informática e de Sistemas
Instituto Superior de Engenharia de Coimbra/Instituto Politécnico de Coimbra



Programação Web

C# – Conceitos Avançados - 3ª Parte

*Departamento de Engenharia Informática e de Sistemas
Instituto Superior de Engenharia de Coimbra/Instituto Politécnico de Coimbra*



Collections

Visão Geral

Colecções em C#

- Existem dois tipos disponíveis:
 - ***Non-generic collections*** - Tipicamente desenvolvidas para trabalhar com tipos *System.Object* e portanto, são *containers* fracamente tipados:
 - ❖ *ArrayList, BitArray, Hashtable, Queue, SortedList, Stack, HybridDictionary, ListDictionary, StringCollection, BitVector32*
 - ❖ Namespaces:
System.Collections, System.Collections.Specialized, System.Collections.ObjectModel, System.Collections.Concurrent

Colecções em C#

- Existem dois tipos disponíveis (cont.):
 - **Generic collections** – Mais seguras que as anteriores, sendo necessário especificar o “tipo” do tipo.
O indicador de qualquer item genérico é o “type parameter” especificado com os <>, como por exemplo, `List<T>`
 - ❖ Namespace: `System.Collections.Generic`
- Todas as classes *collections* implementam a interface *IEnumerable*, portanto os valores da colecção podem ser acedidos usando o *foreach*

Colecções em C# - Genéricas

- *Dictionary<TKey, TValue>*
- *LinkedList<T>*
- *List<T>*
- *Queue<T>*
- *SortedDictionary<TKey, TValue>*
- *SortedSet<T>*
- *Stack<T>*

Collections: Inicialização

- Inicialização com valores específicos:

```
var stringList = new List<string>
{
    "Blabla",
    "BliBli",
};
```

```
var temp = new List<string>();
temp.Add("Blabla");
temp.Add("Blibli");
var stringList = temp;
```

Collections: Inicialização

```
List<int> listaGenerica = new List<int>();
```

```
List<int> listaGenerica = new List<int> { 4,2,5,6,2,1,4,5};
```


Collections: Inicialização

- Inicialização para tipos que permitem vários parâmetros no método:

```
var numberDictionary = new Dictionary<int, string>
{
    { 1, "One" },
    { 2, "Two" },
};
```

```
var temp = new Dictionary<int, string>();
temp.Add(1, "One");
temp.Add(2, "Two");
var numberDictionarynumberDictionary = temp;
```

Collections: Inicialização

```
var dic = new Dictionary<string, int>
{
    ["key1"] = 20,
    ["key2"] = 80
};
```

A partir do C#6

```
var dic = new Dictionary<string, int>();
dict["key1"] = 20;
dict["key2"] = 80
```

Collections: Inicialização

```
public class ClassIndice
{
    public int this[int index] {
        set {
            Console.WriteLine("{0} Atribuido ao indice {1}",
                value, index);
        }
    }
}
```

```
12 Atribuido ao indice 0
15 Atribuido ao indice 1
```

```
static void Main(string[] args) {
    var exemplo = new ClassIndice
    {
        [0] = 12,
        [1] = 15
    };
}
```

Collections: Inicialização

```
class Aluno {  
    public int BI { get; set; }  
    public string Nome {  
        get; set; }  
}
```

```
Aluno aluno= new Aluno {  
    BI = 10,  
    Nome = "Maria"  
};  
Console.WriteLine($"Aluno:{aluno.BI}-{aluno.Nome}");
```

```
List<Aluno> listaGenAlunos = new List<Aluno> {  
    new Aluno { BI=102212232, Nome="Maria" },  
    new Aluno { BI=21222334, Nome="Jose" },  
    new Aluno { BI=102324532, Nome="Nuno" },  
};
```

Inicialização de class com *Collection Initializers*

- Para que uma classe suporte a inicialização do estilo de uma collection, esta deve implementar a interface **IEnumerable** e ter, pelo menos, um **método Add**

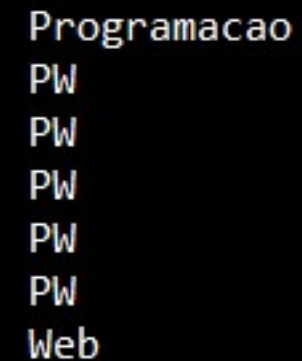
```
var col = new MinhaColecao
{
    "Programacao",
    { "PW", 5 },
    "Web"
};

foreach (var mCol in col)
    Console.WriteLine("{0}", mCol);
```

Inicialização de class com *Collection Initializers*

```
class MinhaColecao : IEnumerable {  
    private IList list = new ArrayList();  
    public void Add(string item) {  
        list.Add(item);  
    }  
    public void Add(string item, int quant) {  
        for (int i = 0; i < quant; i++)  
            list.Add(item);  
    }  
    public IEnumerator GetEnumerator() {  
        return list.GetEnumerator();  
    }  
}
```

```
var col = new MinhaColecao  
{  
    "Programacao",  
    { "PW", 5 },  
    "Web"  
};  
  
foreach (var mCol in col)  
    Console.WriteLine("{0}", mCol);
```



```
Programacao  
PW  
PW  
PW  
PW  
PW  
Web
```

Collection Initializers

- *PropList* é uma propriedade do tipo collection

```
class MinhaClass
{
    public IList<string> PropList { get; set; }
}
```

```
MinhaClass col = new MinhaClass
{
    PropList = new List<string> { "PW", "C#" }
};

foreach (var mCol in col.PropList)
    Console.WriteLine("{0}", mCol);
```

Inicialização de class com *Collection Initializers*

```
var col = new MinhaColecao
{
    "Programacao",
    "Web"
};
foreach (var mCol in col)
    Console.WriteLine("{0}", mCol);
```

```
var col = new MinhaColecao {
    list=new ArrayList {
        "Programacao",
        "Web"
    }
};
foreach (var mCol in col.list)
    Console.WriteLine("{0}", mCol);
```

```
class MinhaColecao : IEnumerable {
    private IList list = new ArrayList();
    public void Add(string item) {
        list.Add(item);
    }
    public void Add(string item, int quant) {
        for (int i = 0; i < quant; i++)
            list.Add(item);
    }
    public IEnumerator GetEnumerator() {
        return list.GetEnumerator();
    }
}
```

```
class MinhaColecao {
    public IList list = new ArrayList();
}
```


Collection Initializers

```
class MinhaClass
{
    public IList<string> PropList { get; private set; }
}
```

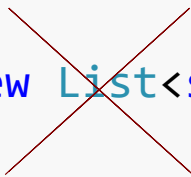
```
MinhaClass col = new MinhaClass
{
    PropList = new List<string> { "PW", "C#" }
};

foreach (var mCol in col.PropList)
    Console.WriteLine("{0}", mCol);
```

Collection Initializers

```
class MinhaClass
{
    public IList<string> PropList { get; private set; }
}
```

```
MinhaClass col = new MinhaClass
{
    PropList = new List<string> { "PW", "C#" }
};
```



Collection Initializers

```
class MinhaClass
{
    public IList<string> PropList { get; private set; }
}
```

```
MinhaClass col = new MinhaClass
```

```
PropList = new List<string> { "PW", "C#" }
```

 `IList<string> MinhaClass.PropList { get; private set; }`

`foreach (v` The property or indexer 'MinhaClass.PropList' cannot be used in this context because the set accessor is inaccessible

```
Console.WriteLine("{0}", mCol);
```

```
Console.WriteLine("{0}", mCol);
```

Collection Initializers

```
class MinhaClass
{
    public IList<string> PropList { get; private set; }
}
```

```
MinhaClass col = new MinhaClass
{
    PropList = { "PW", "C#" }
};
```



Collection Initializers

```
class MinhaClass
{
    public IList<string> PropList { get; private set; }
}
```

```
MinhaClass col = new MinhaClass
{
    PropList = { "PW", "C#" }
};
```



Exceção não processada: System.NullReferenceException: A referência de objecto não foi definida como uma instância de um objecto.

Collection Initializers

```
class MinhaClass
{
    public IList<string> PropList { get; private set; }
}
public MinhaClass()
{
    PropList = new IList<string>();
}
```

```
MinhaClass col = new MinhaClass
{
    PropList = { "PW", "C#" }
};
```

Alguns Metodos úteis da List<T>

- Add(T)
- AddRange(IEnumerable<T>)
- BinarySearch(T)
- Clear()
- Contains(T)
- CopyTo(T[])
- Equals(Object)
- Exists(Predicate<T>)
- Find(Predicate<T>)
- FindAll(Predicate<T>)
- FindIndex(Predicate<T>)
- FindLast(Predicate<T>)
- FindLastIndex(Predicate<T>)
- GetEnumerator()
- IndexOf(T)
- Insert(Int32, T)
- LastIndexOf(T)
- Remove(T)
- RemoveAll(Predicate<T>)
- RemoveAt(Int32)
- Reverse()
- Reverse(Int32, Int32)
- Sort()
- Sort(Comparison<T>)
- ToArray()
- ToString()
- ...

Atenção!

```
var a = new List<int>();  
var b = a;  
a.Add(5);  
  
Console.WriteLine(a.Count);  
Console.WriteLine(b.Count);
```



Qual será o resultado?

1 1

Recordar...

```
var a = new List<int>();  
var b = a;  
a.Add(5);  
b.Add(6);  
b.Add(2);  
Console.WriteLine(a.Count);  
Console.WriteLine(b.Count);
```



Qual será o resultado?

3
3

Mais info...

- **Collections (C#)**

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/collections>

- **C# Quickstart: Collections**

<https://docs.microsoft.com/en-us/dotnet/csharp/quick-starts/arrays-and-collections>

- **Collection<T> Class**

<https://docs.microsoft.com/en-us/dotnet/api/system.collections.objectmodel.collection-1?view=netframework-4.7.2>

- **Collections and Data Structures**

<https://docs.microsoft.com/en-us/dotnet/standard/collections/>

- **Commonly Used Collection Types**

<https://docs.microsoft.com/en-us/dotnet/standard/collections/commonly-used-collection-types>

Delegates

Visão Geral

O que é um Delegate em C# ?



Delegates em C# - Introdução

- Tipos que permitem fazer referência a métodos (*eventos/callbacks*)
- Usado para passar métodos como argumentos a outros métodos
- Um *delegate* é um tipo *referência*
- Pode ser usado para encapsular um método com nome ou anónimo
- Permite **flexibilidade** e **extensibilidade**

Delegates em C#

- **Declaração** de um *delegate* é semelhante à assinatura de um método
- Tem um retorno e qualquer número de parâmetros, de qualquer tipo
- Recorrer à *keyword* *delegate*

```
public delegate void Exemplo(int valor);
```

- Todos os *delegates* derivam da class `System.Delegate`

Delegates em C#

- Quando se atribui um método a um *delegate*, o método deve ter o mesmo tipo de retorno bem como o mesmo tipo de parâmetros;

```
public static void NomeMetodo(string msg) {  
    Console.WriteLine(msg);  
}
```

- Usar o Delegate:

- Declarar

```
public delegate void Declarar(string msg);
```

- Instanciar

```
Declarar xpto=NomeMetodo;
```

- Invocar

```
xpto("mensagem");
```

Delegates em C#

- *Na prática pode-se considerar um delegate como algo similar ao conceito de ponteiros para funções usados nas linguagens C e C++*
- Desse modo, um *delegate* em C#, é semelhante a um ponteiro de função (*a vantagem é que é um ponteiro seguro*).
 - Assim, a utilização de *delegate* permite encapsular a referência a um método dentro de um objeto de delegação.

Delegates em C# - Métodos Anónimos

- Métodos Anónimos
 - A ideia por trás dos métodos anónimos é permitir escrever métodos *inline* no código sem declarar um método formal com nome

Delegates em C# - Exemplo Simples

1. Especificação dos métodos

```
...  
static public string ConverteMaiusculas(string txt)  
{  
    return "Maiuscula = " + txt.ToUpper();  
}  
  
static public string ConverteMinusculas(string txt)  
{  
    return "Minuscula = " + txt.ToLower();  
}  
...
```

Delegates em C# - Exemplo Simples

2. Declaração, instanciação e invocação do *Delegate*

```
delegate string MetodoDel(string texto);

static void Main(string[] args)
{
    string txt = "Programacao Web";

    MetodoDel del = ConverteMaiusculas;
    string msg = del(txt);
    Console.WriteLine("{0}", msg);
    del = ConverteMinusculas;
    Console.WriteLine("{0}", del(txt));
}
```

```
ConverteMaiusculas(string txt)
```

```
return txt.ToUpper();
```

```
ConverteMinusculas(string txt)
```

```
return txt.ToLower();
```

```
MAIUSCULA = PROGRAMACAO WEB
Minuscula = programacao web
```

Delegates em C# - Exemplo Simples

2. Declaração, instanciação e invocação do *Delegate*

```
delegate string MetodoDel(string texto);

static void Main(string[] args)
{
    string txt = "Programacao

    MetodoDel del = ConverteMa
    string msg = del(txt);
    Console.WriteLine("{0}", m
    del = ConverteMinusculas;
    Console.WriteLine("{0}", d
}
```

Forma reduzida
para invocar
o delegate

Poderia ser:
del.Invoke(txt)

las(string txt)

r());

string txt)

PROGRAMACAO WEB

Minuscula = programacao web

Delegates em C# - Exemplo Simples

- Tipo delegate como parâmetro

```
static void DoOperacao(MetodoDel del, string txt)
{
    var ret = del(txt);
    Console.WriteLine(string.Format("> delegate retornou {0}", ret));
}
```

```
static public string ConverteMaiusculas(string txt)
{
    return "Maiuscula = " + txt.ToUpper();
}
static public string ConverteMinusculas(string txt)
{
    return "Minuscula = " + txt.ToLower();
}
```

Delegates em C# - Exemplo Simples

- Tipo delegate como parâmetro

```
static void DoOperacao(MetodoDel del, string txt)
{
    var ret = del(txt);
    Console.WriteLine(string.Format("> delegate retornou {0}", ret));
}

static public static delegate string MetodoDel(string texto);
static public static void Main(string[] args)
{
    return "Maiusculas";
}
static public static void Main(string[] args)
{
    return "Minusculas";
}

static public static void Main(string[] args)
{
    string txt = "Programacao Web";
    DoOperacao(ConvertToUppercase, "Programacao Web");
    DoOperacao(ConvertToLowercase, "Programacao Web");
}
```

Delegates em C# - Exemplo Simples

- Tipo delegate como parâmetro

```
static void DoOperacao(MetodoDel del, string txt)
{
    var ret = del(txt);
}

static void Main(string[] args)
{
    return "Maiu";
}

static public static void Main(string[] args)
{
    string txt = "Programacao Web";
    DoOperacao(ConvertMaiusculas, "Programacao Web");
    DoOperacao(ConvertMinusculas, "Programacao Web");
}
```

> delegate retornou MAIUSCULA = PROGRAMACAO WEB
> delegate retornou Minuscula = programacao web

Delegates em C# - Exemplo Simples

- Instanciar um delegate com um método anónimo

```
delegate string MetodoDel(string texto);
static void Main(string[] args)
{
    string txt = "Programacao Web";
    MetodoDel del2 = delegate (string texto)
    {
        return texto + "|" + texto;
    };
    Console.WriteLine("{0}", del2(txt));
}
```


Delegates em C# - Exemplo Simples

- Instanciar um delegate com um método anônimo

```
delegate string MetodoDel(string texto);  
static void Main(string[] args)  
{  
    string txt = "Programacao Web";  
    MetodoDel del2 = () => "Programacao Web";  
    Console.WriteLine("{0}", del2(txt));  
}
```

Delegates Genéricos

- C# inclui tipos de **bluit-in delegates**!
- Existem 3 tipos de delegate pré-definidos:

- *Func*

```
Func<int, int, int> soma = Soma;
```

- *Action*

```
Action<int> imp = ImprimeConsola;
```

- *Predicate*

```
Predicate<string> estaMai = EstaMaiuscula;
```

Delegates Genéricos

- Estes Delegates “**bluit-in**” existem para facilitar a vida ao programador!
- Poderiam sempre ser implementados a partir de um *Delegate* Genérico
- Na prática correspondem às situações mais vulgares, mais comuns, de utilização de Delegates e então em vez de o programador estar a “criar” um a partir de um Genérico, a ideia é usar um destes que já está, digamos, que “feito”

Delegate Func

▪ *Func*

- É um **tipo genérico de delegate**
- Tem **zero ou mais parâmetros de entrada e um parâmetro de saída**
- O último parâmetro é considerado o parâmetro de saída
- Um delegate do tipo Func pode incluir 0 a 16 parâmetros de entrada de tipos diferentes
- Deve incluir um parâmetro **out** para resultado

Delegate Func

```
public delegate int Operacao(int i, int j);
class Program
{
    static int Soma(int x, int y)
    {
        return x + y;
    }
    static void Main(string[] args)
    {
        Operacao soma = Soma;
        int resultado = soma(10, 10);
        Console.WriteLine(resultado);
    }
}
```

```
class Program
{
    static int Soma(int x, int y)
    {
        return x + y;
    }
    static void Main(string[] args)
    {
        Func<int, int, int> soma = Soma;
        int resultado = soma(10, 10);
        Console.WriteLine(resultado);
    }
}
```

Delegate Func

- **Func** com método **Anónimo**

```
Func<int> geraAleatorio = delegate()  
{  
    Random rnd = new Random();  
    return rnd.Next(1, 100);  
};
```

Delegate Action

■ Action

- Do mesmo género que o delegate Func, mas o Action não devolve resultado
- Em termos conceptuais, pode pensar nos “retornos” *void* de funções (atenção é um Delegate não uma função)

```
Action<int> imp = ImprimeConsola;
```

```
Action<int> imp = new Action<int>(ImprimeConsola);
```

Delegate Action

```
public delegate void Imprime(int val);

static void ImprimeConsola(int i)
{
    Console.WriteLine(i);
}

static void Main(string[] args)
{
    Imprime imp = ImprimeConsola;
    imp(10);
    imp(20);
    imp(30);
}
```

```
static void ImprimeConsola(int i)
{
    Console.WriteLine(i);
}

static void Main(string[] args)
{
    Action<int> imp = ImprimeConsola;
    imp(10);
    imp(20);
    imp(30);
}
```


Delegate Action – Método Anónimo

```
static void ImprimeConsola(int i)
{
    Console.WriteLine(i);
}

static void Main(string[] args)
{
    Action<int> imp = ImprimeConsola;
    imp(10);
    imp(20);
    imp(30);
}
```

```
static void Main(string[] args)
{
    Action<int> imp = delegate (int i)
    {
        Console.WriteLine(i);
    };
    imp(10);
    imp(20);
    imp(30);
}
```

Delegate Predicate

- *Predicate*

- Representa um método que contém um conjunto de critérios e verifica se os parâmetros passados cumprem esse critério ou não -> caso “cumpram” retorna “TRUE”; caso contrário retorna “FALSE”
- Deve ter um parâmetro de entrada e retorna um booleano – verdadeiro ou falso

Delegate Predicate

```
static bool EstaMaiuscula(string str)
{
    return str.Equals(str.ToUpper());
}

static void Main(string[] args)
{
    Predicate<string> estaMai = EstaMaiuscula;

    bool result = estaMai("programacao");
    Console.WriteLine(result);
    result = estaMai("WEB");
    Console.WriteLine(result);
}
```

false
true

Delegate Predicate – Método Anónimo

```
static void Main(string[] args)
{
    Predicate<string> estaMai = delegate (string s)
    {
        return s.Equals(s.ToUpper());
    };

    bool result = estaMai("programacao");
    Console.WriteLine(result);
    result = estaMai("WEB");
    Console.WriteLine(result);
}
```

false
true

Func vs Action vs Predicate

- Para referenciar um método que tem apenas um parâmetro e retorna **void**, deve-se usar antes o *delegate genérico Action<T>*
- *Predicate<T>* é também uma forma de *Func* mas retorna sempre um **bool**
 - Forma de especificar um determinado critério;
 - Comporta-se do mesmo modo que **Func<T, bool>**

Delegates Encadeados

- *Delegates **Multicast*** permitem guardar a referência para mais de um método
- Para adicionar uma referência, pode-se usar o operador +=
- Os métodos são executados pela ordem que foram adicionados
- Faz sentido em métodos void

Delegates Encadeados: Exemplo

■ Considere as classes:

```
public class Bebe
{
    public string Nome { get; set; }
    public Bebe(string n)
    {
        Nome = n;
    }

    public static Bebe Nasce(string nome)
    {
        return new Bebe(nome);
    }

    public void GravaInfo() {}
}
```

```
public class BebeExames
{
    public void Peso(Bebe bebe)
    {
        Console.WriteLine("Pesar Bébé!");
    }
    public void Audicao(Bebe bebe)
    {
        Console.WriteLine("Efetuar exame de audição");
    }
    public void FrequenciaCardiaca(Bebe bebe)
    {
        Console.WriteLine("Obter frequência cardíaca");
    }
}
```

Delegates Encadeados: Exemplo

```
public class ProcessoDoBebe
{
    public void Nascimento(string nome)
    {
        var bebe = Bebe.Nasce(nome);
        var exames = new BebeExames();
        exames.Peso(bebe);
        exames.FrequenciaCardiaca(bebe);
        exames.Audicao(bebe);
        bebe.GravaInfo();
    }
}
```

```
ProcessoDoBebe processo = new ProcessoDoBebe();
processo.Nascimento("Nuno");
```

Pesar Bébé
Obter frequência cardíaca
Efetuar exame de audicao

*Imaginemos que é efetuado
o release, e queremos
adicionar novo exame?*

Delegates Encadeados: Exemplo

```
public class ProcessoDoBebe
{
    public void Nascimento(string nome)
    {
        var bebe = Bebe.Nasce(nome);
        var exames = new BebeExames();
        exames.Peso(bebe);
        exames.FrequenciaCardiaca(bebe);
        exames.Audicao(bebe);
        bebe.GravaInfo();
    }
}
```

```
public delegate void BebeExamesDel(Bebe bebe);
public void Nascimento(string nome, BebeExamesDel exame)
{
    var bebe = Bebe.Nasce(nome);
    exame(bebe);
    bebe.GravaInfo();
}
}
```

```
var processo = new ProcessoDoBebe();
var bebeExames = new BebeExames();
ProcessoDoBebe.BebeExamesDel exames = bebeExames.Peso;
exames += bebeExames.Audicao;
exames += bebeExames.FrequenciaCardiaca;
exames += bebeExames.Peso;

processo.Nascimento("Nuno", exames);
```

Pesar Bébé
Obter frequência cardíaca
Efetuar exame de audição
Pesar Bébé

Delegates Encadeados: Exemplo

E adicionar novo exame inexistente na classe BebeExames?

```
static void Main()
{
    var processo = new ProcessoDoBebe();
    var bebeExames = new BebeExames();
    ProcessoDoBebe.BebeExamesDel exames = bebeExames.Peso;
    exames += bebeExames.Audicao;
    exames += bebeExames.FrequenciaCardiaca;
    exames += bebeExames.Peso;
    exames += bebeExames.Ictericia;

    processo.Nascimento("Nuno", exames);
}
```

```
static void Ictericia(Bebe bebe)
{
    Console.WriteLine("Verificar Ictericia!");
}
```

Delegates Encadeados: Exemplo

```
public class ProcessoDoBebe
{
public delegate void BebeExamesDel(Bebe bebe);
public void Nascimento(string nome, Action<Bebe> exame)
{
    var bebe = Bebe.Nasce(nome);
    exame(bebe);
    bebe.GravaInfo();
}
}
```

```
var processo = new ProcessoDoBebe();
var bebeExames = new BebeExames();
ProcessoDoBebe.BebeExamesDel exames = bebeExames.Peso;
Action<Bebe> exames = bebeExames.Peso;
exames += bebeExames.Audicao;
exames += bebeExames.FrequenciaCardiaca;
exames += bebeExames.Peso;

processo.Nascimento("Nuno", exames);
```

**Recorrendo
ao Action**

Pesar Bébé
Obter frequência cardíaca
Efetuar exame de audicao
Pesar Bébé

Delegates em C# - Exemplo com Lambda

- Instanciar um delegate com uma **expressão lambda**

```
delegate string MetodoDel(string texto);  
static void Main(string[] args)  
{  
    string txt = "Programacao Web";  
  
    MetodoDel del3 = texto=>texto.ToUpper();  
  
    Console.WriteLine("{0}", del3(txt));  
}
```

PROGRAMACAO WEB

Lambda

O que é uma Expressão Lambda ?



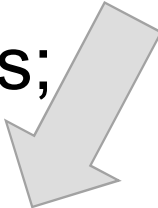
Operador Lambda =>

▪ OPERADOR LAMBDA =>

- Este operador pode ser usado de duas maneiras no C#:
 1. Como operador lambda numa expressão lambda: separa as variáveis de entrada do lado esquerdo do corpo lambda do lado direito
 2. Numa definição de corpo da expressão, em que separa um nome de membro da implementação do membro.
- O operador => tem a mesma precedência que o operador de atribuição = e é associativo-à-direita

O que é uma Expressão Lambda?

- As expressões *lambda* são expressões embutidas semelhantes aos métodos anónimos, mas mais flexíveis;
- Método anónimo
 - Sem nome
 - Sem modificador de acesso (como público ou privado)
 - Não retorna valor
- Usado como um atalho para inicialização de um *delegate*



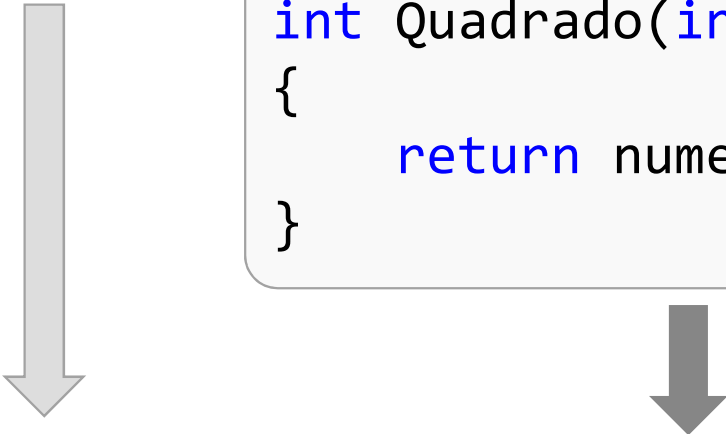
Permitem escrever menos código e atingir o mesmo objetivo

Expressão Lambda

- Sintaxe:

argumentos => expressão

- Exemplo:



```
int Quadrado(int numero)
{
    return numero * numero;
}
```

```
int Quadrado(int numero)=> numero * numero;
```


Lambda

- Com argumentos

$x \Rightarrow \dots$

- Sem Argumentos

$() \Rightarrow \dots$

- Com vários argumentos

$(x, y) \Rightarrow \dots$

Lambda: Exemplo

```
public delegate int OperacaoComInteiro(int input);

static void Main(string[] args)
{
    OperacaoComInteiro dobro = x => x * 2;
    OperacaoComInteiro quadrado = x => x*x;

    Console.WriteLine("    ->{0}", dobro(5));
    Console.WriteLine("    ->{0}", quadrado(5));
}
```

->10
->25

Lambda: Exemplo com um Argumento

```
public delegate int DobroInteiro(int input);  
DobroInteiro dobro = x => x * 2;
```



```
public delegate int DobroInteiro(int input);  
DobroInteiro dobro = delegate (int x) {  
    return x * 2;  
};
```

Lambda: Exemplo vários argumentos

```
public delegate int MultiplicaInteiros(int n1, int n2);  
  
MultiplicaInteiros mult = (x,y) => x*y;  
  
Console.WriteLine("->{0}", mult(2,7));
```

Lambda: Exemplo sem argumentos

```
public delegate string EscreveMensagem();  
  
EscreveMensagem msg = () => "Programacao";  
EscreveMensagem msg1 = () => "Web";  
EscreveMensagem msg2 = () => "C#";  
  
Console.WriteLine(" {0} | {1} | {2} ", msg(), msg1(), msg2());
```

Programacao | Web | C#

Várias instruções numa Exp. Lambda

```
public delegate void MultiplicaInteiros(int n1, int n2);  
MultiplicaInteiros mult = (x, y) =>  
{  
    int result = x * y;  
    Console.WriteLine("Resultado Multiplicação = {0}", result);  
};  
mult(5, 1);  
mult(5, 2);  
mult(5, 3);  
mult(5, 4);  
mult(5, 5);
```

```
Resultado Multiplicação = 5  
Resultado Multiplicação = 10  
Resultado Multiplicação = 15  
Resultado Multiplicação = 20  
Resultado Multiplicação = 25
```

Outros formas de utilização do Lambda

- Passar uma expressão lambda como parâmetro de um método

```
List<int> lista1 = new List<int> { 1, 2, 6, 5, 4, 3, 2, 7, 8, 5, 24, 42 };  
List<int> lista2 = new List<int> { 12, 33, 44, 2, 5, 67, 86, 4, 3, 22, 56, 77 };
```

```
List<int> listaResult = lista1.FindAll(x => x >= 24);
```

```
List<int> listaResult = lista1.FindAll()  
List<int> listaResult2 = lista1.FindAll(Predicate<int> match)  
(int x in list) {  
    Console.WriteLine(x);  
    return x >= 24;  
}).WriteLine("-----");
```

List<int> List<int>.FindAll(Predicate<int> match)
Retrieves all the elements that match the conditions defined by the specified predicate.
match: The Predicate<in T> delegate that defines the conditions of the elements to search for.

Outros formas de utilização do Lambda

- Passar uma expressão lambda como parâmetro de um método

```
List<int> lista1 = new List<int> { 1, 2, 6, 5, 4, 3, 2, 7, 8, 5, 24, 42 };  
List<int> lista2 = new List<int> { 12,33,44,2,5,67,86,4,3,22,56,77 };
```

```
List<int> listaResult=lista1.FindAll(x => x >= 24);
```

```
List<int> listaResult2 = lista2.FindAll(x => x >= 24);
```

Lambda atua como um *Predicate* que garante que apenas os números maiores que 24 são retornados

Expressão Lambda Simples - Exemplo

```
static void Main(string[] args)
{
    Func<int, int> dobro = i => i * 2;
    Console.WriteLine(dobro(5));
}
```

10

```
public delegate int CalculaDobro(int n);
static void Main(string[] args)
{
    CalculaDobro dobro = i => i * 2;

    Console.WriteLine(dobro(5));
}
```

Expressão Lambda Simples - Exemplo

```
static void Main(string[] args)
{
    Func<int, int> dobro= i => i * 2;
    Func<int, int, int> soma = (i, j) => i + j;

    Console.WriteLine(dobro(5));
    Console.WriteLine(Dobro(5));

    Console.WriteLine(soma(11, 20));
    Console.WriteLine(Soma(11, 20));
}
```



Func

É um tipo genérico de delegate

Existem 3 tipos:

- *Func*
- *Action*
- *Predicate*

Expressão Lambda Simples - Exemplo

```
static void Main(string[] args)
{
    Func<int, int> dobro= i => i * 2;
    Func<int, int, int> soma = (i, j) => i + j;

    Console.WriteLine(dobro(5));
    Console.WriteLine(Dobro(5));

    Console.WriteLine(soma(11, 20));
    Console.WriteLine(Soma(11, 20));
}
```

10
10
31
31

```
static int Dobro(int i)
{
    return i * 2;
}
static int Soma(int i, int j)
{
    return i + j;
}
```

Expressão Lambda Simples - Exemplo

```
static void Main(string[] args)
{
    Func<string, string> convMaiuscula = str => str.ToUpper();

    string[] palavras = { "programacao", "web", "disciplina" };

    IEnumerable<string> selPal = palavras.Select(convMaiuscula);

    foreach (string palavra in selPal)
        Console.WriteLine(palavra);
}
```

**PROGRAMACAO
WEB
DISCIPLINA**

Func vs Action vs Predicate com Lambda

```
Predicate<string> predicado = s => s.StartsWith("a");  
Func<string, bool> func = s => s.StartsWith("a");
```

```
string str = "programacao web";  
Console.WriteLine(predicado(str));  
Console.WriteLine(predicado("a"+str));
```

false
true

Expressão Lambda

- Uso do delegado `Func<T,TResult>` com métodos anónimos

```
Func<string, string> converte = delegate (string s)
{
    return s.ToUpper();
};
```

```
string str = "programacao web";
Console.WriteLine(converte(str));
```

14
30

Expressão Lambda

```
Func<int, string> dobroesomar10 = i => {  
    var dobro = 2 * i;  
    var soma = 10 + dobro;  
    return $">{soma}<";  
};  
  
Console.WriteLine("{0}", dobroesomar10(2));  
Console.WriteLine("{0}", dobroesomar10(10));
```

14
30

Lambda: Exemplo

- Criação de um array de Alunos

```
Aluno[] alunos = {  
    new Aluno() { Numero = 1, Nome= "Jose Antunes", Idade= 18 },  
    new Aluno() { Numero = 2, Nome = "Filipa Moreira", Idade = 21 },  
    new Aluno() { Numero = 3, Nome = "Cristina Fria", Idade = 25 },  
    new Aluno() { Numero = 4, Nome = "Dinis Campos" , Idade = 20 },  
    new Aluno() { Numero = 5, Nome = "Soaraia Goncalves" , Idade = 31 },  
    new Aluno() { Numero = 6, Nome = "Lena Pinheiro", Idade = 17 },  
    new Aluno() { Numero = 7, Nome = "Filipa Goncalves", Idade = 18 },  
};
```

```
public class Aluno  
{  
    public int Numero { get; set; }  
    public string Nome { get; set; }  
    public int Idade { get; set; }  
}
```


Lambda: Exemplo

```
static bool MaioresIdade(Aluno aluno)
{
    return aluno.Idade >= 18;
}
```



```
alunos.FindAll(aluno => aluno.Idade >= 18);
alunos.FindAll(a => a.Idade >= 18);
```

Muito usados
em consultas
LINQ

Lambda: Exercício

- Considerando o array de alunos, especifique uma expressão lambda que permita obter a lista de alunos com a letra F no nome:

```
Aluno[] alunos = {  
    new Aluno() { Numero = 1, Nome= "Jose Antunes", Idade= 18 },  
    new Aluno() { Numero = 2, Nome = "Filipa Moreira", Idade = 21 },  
    new Aluno() { Numero = 3, Nome = "Cristina Fria", Idade = 25 },  
    new Aluno() { Numero = 4, Nome = "Dinis Campos", Idade = 20 },  
    new Aluno() { Numero = 5, Nome = "Soaraia Goncalves", Idade = 31 },  
    new Aluno() { Numero = 6, Nome = "Lena Pinheiro", Idade = 17 },  
    new Aluno() { Numero = 7, Nome = "Filipe Cruz", Idade = 19 },  
};
```



Lambda: Exemplo

- Considerando o array de alunos, especifique uma expressão lambda que permita obter a **lista de alunos com a letra F** no nome:

```
var alunosF = alunos.Where(???)
```

```
var alunosF = alunos.Where(  
    (e) => {  
        return (e.Nome.Contains('F'));  
    } );
```



Lambda: Exemplo

- E se apenas a primeira letra começar por F?

```
var alunosF = alunos.Where(???)
```



```
var alunosF = alunos.Where(  
    (e) => {  
        return (e.Nome.StartsWith("F"));  
    } );
```

Lambda: Exemplo

- E apresentar os nomes dos alunos que tenham a vogal a na primeira palavra do nome?

```
var alunosF = alunos.Where(???)
```



```
var alunosF = alunos.Where(  
    (e) => {  
        return (e.Nome.Split(' ').First().Contains("a"));  
    }  
);
```

C# - Extension Methods



Extension Methods


■ *Extension Methods*

- Permitem adicionar métodos a uma *class* existente sem alterar o seu código fonte ou criar uma nova *class* que derive dela
 - *Útil principalmente em situações em que não é possível alterar o código fonte para um dado tipo*
- Podem ser especificados para tipos do sistema, tipos definidos por terceiros, tipos próprios
- Deve-se adicionar um método estático a uma classe estática

Extension Methods

```
static void Main(string[] args)
{
    string texto = "PROGRAMACAO WEB";
    PWString.AddPW(texto);
}

public static class PWString:string
{
    public string AddPW(string x)
    {
        return x + "+PW";
    }
}
```



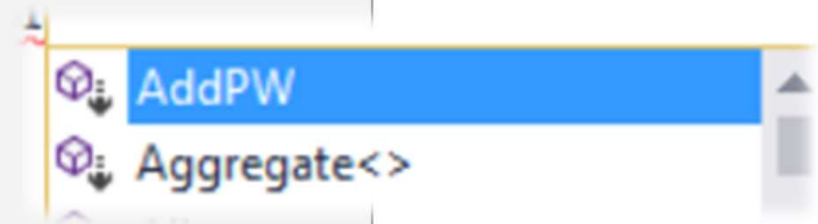
Extension Methods

```
public static class PWStringExtensions
{
    public static string AddPW(this String x)
    {
        return x + " +PW";
    }
}
class Program
{
    static void Main(string[] args)
    {
        string texto = "PROGRAMACAO WEB";
        var x = texto.AddPW();
        Console.WriteLine(x);
    }
}
```



Extension Methods

```
public static class PWExtensions
{
    public static string AddPW(this String x)
    {
        return x + " +PW";
    }
}
class Program
{
    static void Main(string[] args)
    {
        string texto = "PROGRAMACAO WEB";
        var x = texto.AddPW();
        Console.WriteLine(x);
    }
}
```



Tipo Nullable

Operador Null-Coalescing



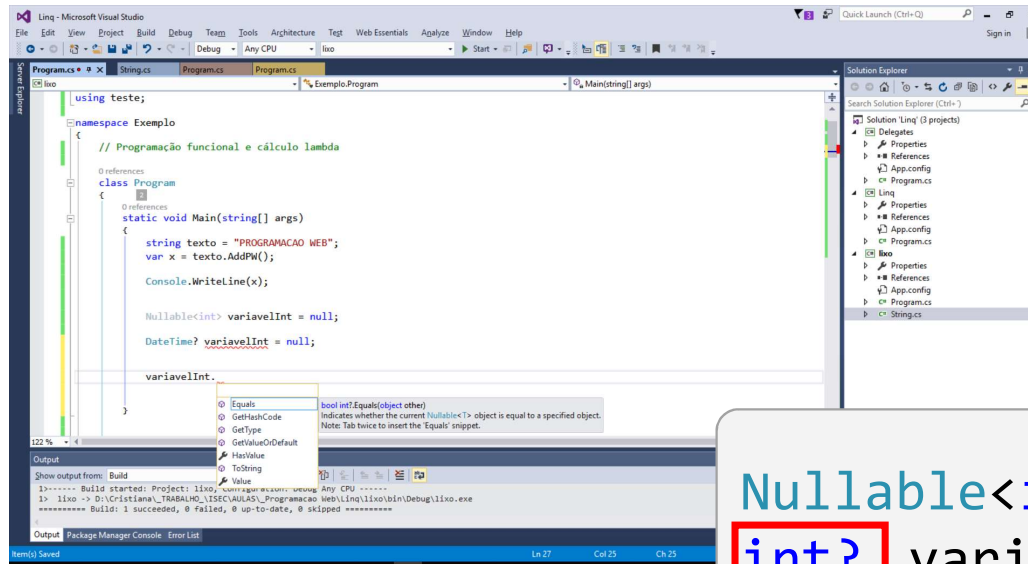
Tipo *Nullable*

- Tipo *Nullable* – *Revisitando estes tipos*
 - Em C# tipos que sejam do tipo valor não podem ter o valor null

Cannot convert null to 'int' because it is a non-nullable value type

```
int variavelInt = null;  
  
bool variavelBool = null;  
  
DateTime variavelDateTime = null;
```

Tipo Nullable



GetValueOrDefault
Value
HasValue

```
Nullable<int> variavelInt = null;  
int? variavelInt = null;
```

```
DateTime? variavelDate = null;  
variavelDate.
```

Tipo *Nullable*

```
int? varInt = null;  
int varInt2;
```

```
Console.WriteLine(varInt.GetValueOrDefault());  
Console.WriteLine(varInt.HasValue);  
Console.WriteLine(varInt.Value);
```



```
C:\WINDOWS\system32\cmd.exe  
0  
False  
  
Exceção não processada: System.InvalidOperationException: O objecto que  
pode ser nulo tem de ter um valor.  
    em System.ThrowHelper.ThrowInvalidOperationException(ExceptionResource  
e resource)  
    em System.Nullable`1.get_Value()  
    em Exemplo.Program.Main(String[] args) em D:\Cristiana\_TRABALHO\_ISE  
C\AULAS\_Programacao Web\Linq\lixo\Program.cs:line 20  
Press any key to continue . . .
```

Tipo *Nullable*

```
int? varInt = null;  
int varInt2;  
  
varInt2=varInt;  
  
varInt2=varInt.GetValueOrDefault();  
  
varInt = varInt2;  
  
varInt2=0;  
varInt = varInt2;
```



Operador *Null-Coalescing*

- Permite especificar um valor por omissão se o operando à esquerda for nulo

```
DateTime? data = null;  
DateTime data1;  
string str = null;  
data1 = data ?? DateTime.Now;  
Console.WriteLine("string = " + (str ?? " Sem valor!"));
```


Operador *Null-Coalescing*

```
int? varInt=null;  
int varInt2;  
  
varInt2 = varInt ?? 10;  
  
int? varInt3 = varInt2;  
  
Console.WriteLine(varInt2+" "+varInt3);
```



Operador *Null-Coalescing*

```
int? varInt = null;  
int? varInt2=null;  
  
int varInt3 = varInt2 ?? varInt ?? 10;
```

```
IEnumerable<Aluno> minhaLista = GetListaAlunos();  
var a1 = minhaLista.SingleOrDefault(x => x.Id == 2) ??  
                                         new Aluno { Id = 2 };  
Console.WriteLine(a1);
```