

Ambientes Virtuais de Execução

Delegates

Callbacks

- Um **callback** é uma referência para um código executável, que é passada como argumento a outro código.

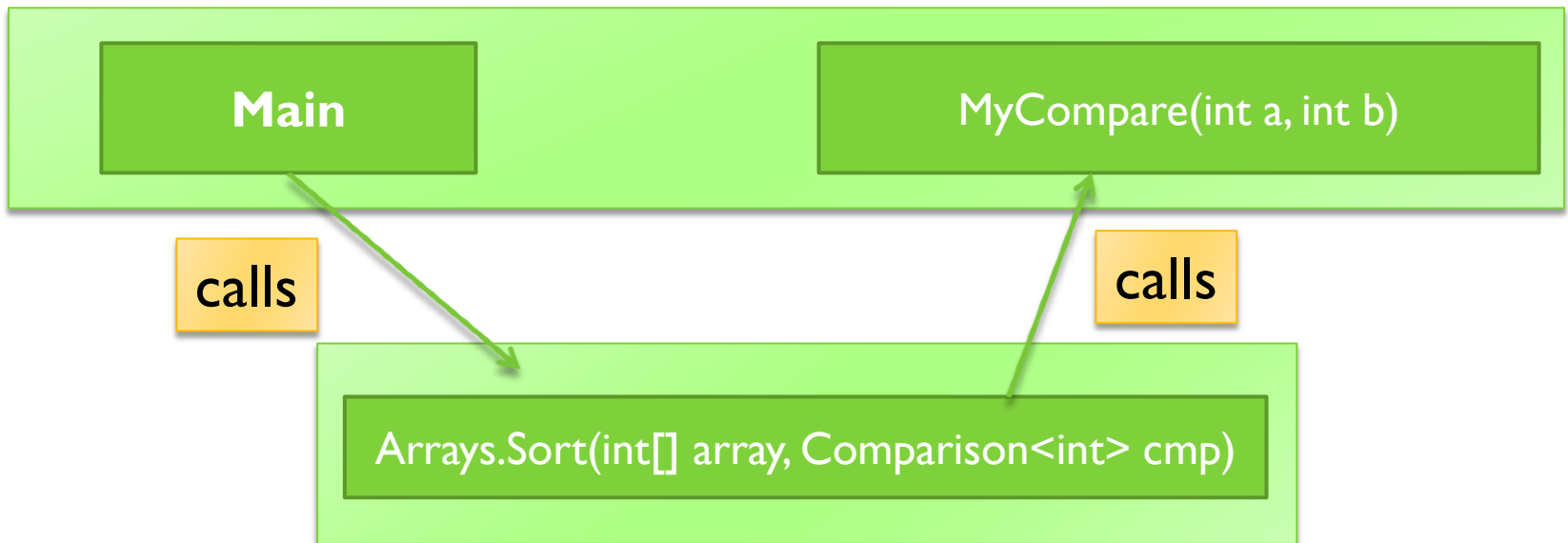


Imagem adaptada da wikipedia

Exemplo de Callbacks em Java

```
import java.util.Comparator;
import java.util.Arrays;
public class Sorting {
    public static void main(String[] args){

        Comparator<Integer> c=new Comparator<Integer>(){
            public int compare(Integer i1, Integer i2){ return i1-i2; }
        };
        Integer[] vals={2,34,5,6,7};
        Arrays.sort(vals,c);
        for (int i =0;i<vals.length;i++) {
            System.out.println(vals[i]);
        }
    }
}
```

Callbacks em C/C++

```
typedef int (*Comparator)(const void *, const void *);
```

```
int compareInts(const void*i1, const void *i2) {  
    return *((int *) i1) - *((int *) i2);  
}
```

```
void testQSort() {  
    int  vals[] = { 2, 8 , 13, 5, 4 };  
    int nelems = sizeof(vals)/sizeof(int);  
    qsort(vals, nelems, sizeof(int), compareInts);  
    for (int i=0; i < nelems; ++i) {  
        Console::WriteLine(vals[i]);  
    }  
}
```

Callbacks em C#

```
using System;
using System.Collections.Generic;
public class Sorting {

    private static int MyComparison(int i1, int i2) {
        return i1 - i2;
    }

    public static void Main(){
        int[] vals={2,34,5,6,7};
        Array.Sort<int>(vals,MyComparison);
        foreach (int i in vals) {
            Console.WriteLine(i);
        }
    }
}
```

Callbacks em C#

```
Array.Sort<int>(vals, MyComparison);
```

Sort<T>(T[], Comparison<T>)

Comparison<T>
é um Delegate

```
public delegate int Comparison<in T>( T x, T y )
```

Delegates na CLI

- ▶ Mecanismo de suporte a *callbacks* fornecido pelo *Runtime*
 - ▶ *Um delegate indica a assinatura de um método callback*
- ▶ Generalização *type-safe* do conceito de ponteiro para função em C/C++, que permite usar como *callbacks* métodos estáticos ou métodos de instância.

Definição de um tipo delegate em C#

```
public delegate int BinaryOp(int x, int y);
```

```
.class public auto ansi sealed BinaryOp extends [mscorlib]System.MulticastDelegate {  
    .method public hidebysig specialname rtspecialname instance void .ctor(  
        object 'object', native int 'method') runtime managed  
    {  
    } // end of method BinaryOp::.ctor  
  
    .method public hidebysig newslot virtual instance int32 Invoke(int32 x, int32 y)  
        runtime managed { } // end of method BinaryOp::Invoke  
  
    .method public hidebysig newslot virtual instance class  
        [mscorlib]System.IAsyncResult  
        BeginInvoke(int32 x, int32 y, class [mscorlib]System.AsyncCallback callback,  
            object 'object') runtime managed  
    { } // end of method BinaryOp::BeginInvoke  
  
    .method public hidebysig newslot virtual instance int32  
        EndInvoke(class [mscorlib]System.IAsyncResult result) runtime managed  
    { } // end of method BinaryOp::EndInvoke  
  
} // end of class BinaryOp
```


Delegates: tipo gerado pelo compilador

- ▶ Em C#, a palavra reservada **delegate** define um novo tipo
- ▶ Quando o compilador de C# processa um tipo delegate, gera automaticamente uma sealed class
 - ▶ deriva de `System.MulticastDelegate` (que por sua vez derivada de `System.Delegate`).
- ▶ A classe gerada define 3 métodos:
 - ▶ `Invoke()`
 - ▶ Usado para invocar de um modo síncrono cada método mantido pelo objecto delegate
 - O *caller* tem de esperar que a *call* termine para continuar a executar.
 - O método `Invoke()` não é chamado explicitamente
 - ▶ `BeginInvoke()` e `EndInvoke()`
 - ▶ permitem invocar o método actual assincronamente numa *thread* separada.

Exemplo 1

```
public delegate int BinaryOp(int x, int y);

public class SimpleMath{
    public static int Add1(int x, int y) { return x + y; }
    public static int Subtract1(int x, int y) {return x - y; }
}

class Program{
    static void Main(string[] args){
        BinaryOp b = new BinaryOp(SimpleMath.Add1);
        // Invoca o método Add1() method indirectamente, usando um objecto delegate
        Console.WriteLine("10 + 10 is {0}", b(10, 10) );
        Console.WriteLine("10 - 10 is {0}",
                           SimpleMath.Subtract1(10, 10));
    }
}
```

Exemplo 1 – Add método estático

```
//...
public delegate int BinaryOp(int x, int y);
public class SimpleMath{
    public static int Add1 (int x, int y) { return x + y; }
    public static int Subtract1 (int x, int y) {return x - y; }
}
class Program{
    static void DisplayDelegateInfo(Delegate delObj){
        foreach (Delegate d in delObj.GetInvocationList()){
            Console.WriteLine("Method Name: {0}", d.Method);
            Console.WriteLine("Type Name: {0}", d.Target);
        }
    }
    static void Main(string[] args){
        BinaryOp b = new BinaryOp(SimpleMath.Add1);
        Console.WriteLine("10 + 10 is {0}", b(10, 10) );
        DisplayDelegateInfo(b); }
}
```

O nome não aparece?

10 + 10 is 20

10 - 10 is 0

Method Name: Int32 Add1(Int32,Int32)

Type Name:

Exemplo 2 – Add método não estático

```
//...
public class SimpleMath{
    public int Add2(int x, int y) { return x + y; }
    public int Subtract2(int x, int y) {return x - y; }
}
class Program{
    static void DisplayDelegateInfo(Delagate delObj){
        foreach (Delegate d in delObj.GetInvocationList()){
            Console.WriteLine("Method Name: {0}", d.Method);
            Console.WriteLine("Type Name: {0}", d.Target);
        }
    }
    static void Main(string[] args){
        BinaryOp b = new BinaryOp((new SimpleMath()).Add2);
        Console.WriteLine("10 + 10 is {0}", b(10, 10) );
        DisplayDelegateInfo(b); }
}
```

O nome não aparece?

10 + 10 is 20

10 - 10 is 0

Method Name: Int32 Add2(Int32,Int32)

Type Name: SimpleMath



Cadeia de Delegates (Multicasting)

```
using System;

public delegate void BinaryOp3(int x, int y);

public class SimpleMath{
    public void Add3 (int x, int y) {Console.WriteLine("{0} + {1} is {2}", x,y, x+y);}
    public void Subtract3 (int x,int y){Console.WriteLine("{0} - {1} is {2}", x,y, x-y);}
}

class Program{
    static void DisplayDelegateInfo(Delagate delObj){
        foreach (Delegate d in delObj.GetInvocationList()){
            Console.WriteLine("Method Name: {0}", d.Method);
            Console.WriteLine("Type Name: {0}", d.Target);
        }
    }

    static void Main(string[] args){
        SimpleMath s=new SimpleMath();
        BinaryOp3 b1 = s.Add3;
        BinaryOp3 b2 = s.Subtract3;
        BinaryOp3 b3 = b1 + b2;
        b3(10, 10);
        DisplayDelegateInfo(b3);
    }
}
```

Um delegate
pode chamar
mais do que
um método
quando
invocado

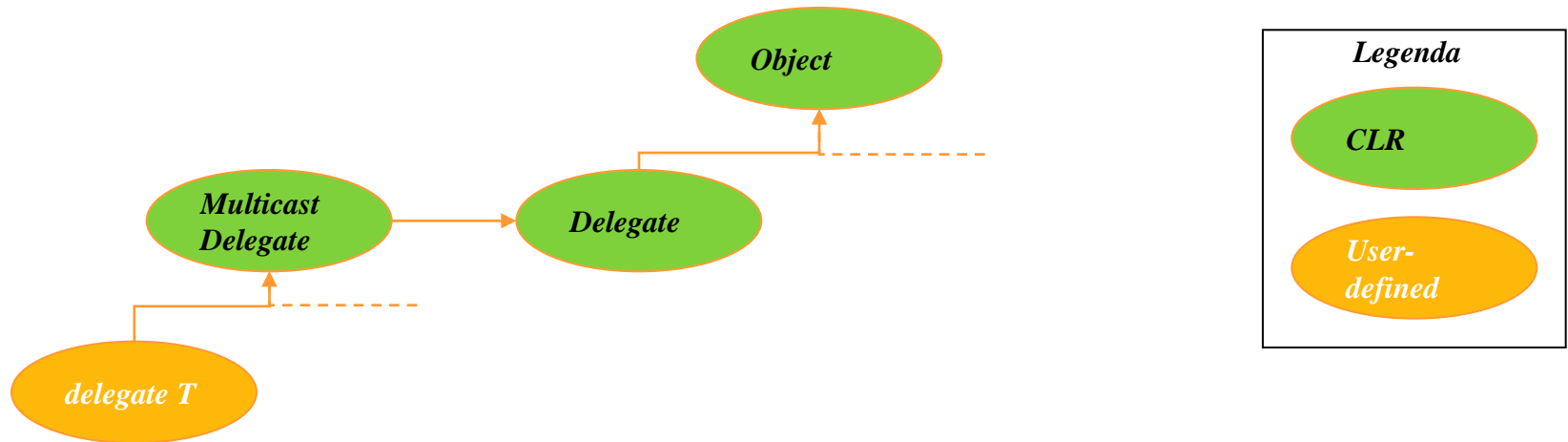
```
10 + 10 is 20
10 - 10 is 0
Method Name: Void Add3(Int32,Int32)
Type Name: SimpleMath
Method Name: Void Subtract3(Int32,Int32)
Type Name: SimpleMath
```



Operadores += e -=

- ▶ instâncias de *delegates* são imutáveis
- ▶ Duas instâncias podem ser combinadas, dando origem a uma terceira instância
 - ▶ O operador += invoca o método `Delegate.Combine()`
 - ▶ Sintaxe simplificada em C#
 - ▶ O método `Delegate.Combine()` pode ser invocado directamente.
 - ▶ A chamada do método `Invoke()` da terceira instância resulta na chamada dos métodos associados às duas instâncias originais
- ▶ Método `Delegate.Remove()` (operador -= em C#) realiza a remoção

Delegates no CLI: A classe Delegate



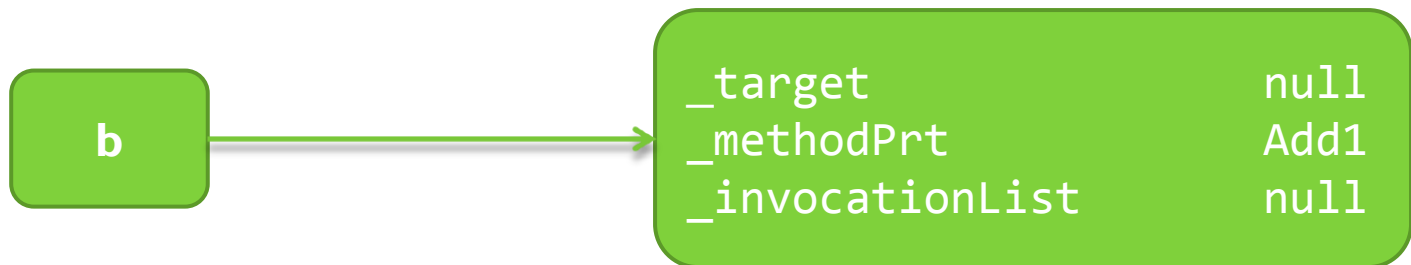
```
public abstract class Delegate {  
    ...  
    public static Delegate Combine(Delegate d1, Delegate d2);  
    public static Delegate Remove(Delegate source, Delegate d);  
    public virtual Delegate[] GetInvocationList();  
    public MethodInfo Method { get; }  
    public Object Target { get; }  
    public static Boolean operator==(Delegate d1, Delegate d2);  
    public static Boolean operator!=(Delegate d1, Delegate d2);  
}
```

Campos não públicos da classe MultiDelegate

Campo	Tipo	Descrição
_target	System.Object	Quando o objecto delegate encapsula um método estático , este campo fica a null . Quando o objecto delegate encapsula um método de instância , este campo refere-se ao objecto que deve ser operado quando o callback método é invocado.
_methodPtr	System.IntPtr	Um inteiro interno que o CLR utiliza para identificar o método que é para ser invocado
_invocationList	System.Object	Ou está a null ou refere-se a um array quando se constroi um multicast delegate

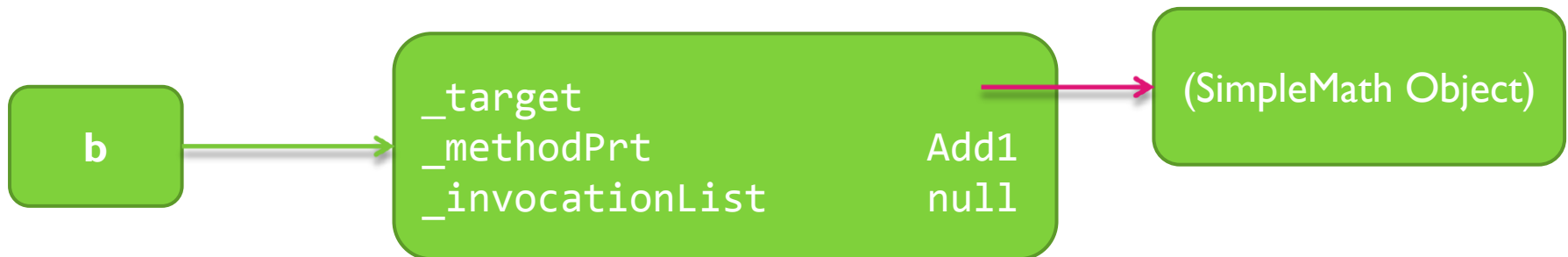
Exemplo 1 (continuação)

```
//...
public delegate int BinaryOp(int x, int y);
public class SimpleMath{
    public static int Add1(int x, int y) { return x + y; }
    //...
}
class Program{
    static void Main(string[] args){ BinaryOp b = new BinaryOp(SimpleMath.Add1);
    //...
}}
```



Exemplo 2 (continuação)

```
//...
public delegate int BinaryOp(int x, int y);
public class SimpleMath{
    public int Add2(int x, int y) { return x + y; }
    //...
}
class Program{
    static void Main(string[] args){
        BinaryOp b = new BinaryOp((new SimpleMath()).Add2);
        //...
    }
}
```



Exemplo 3 (continuação)

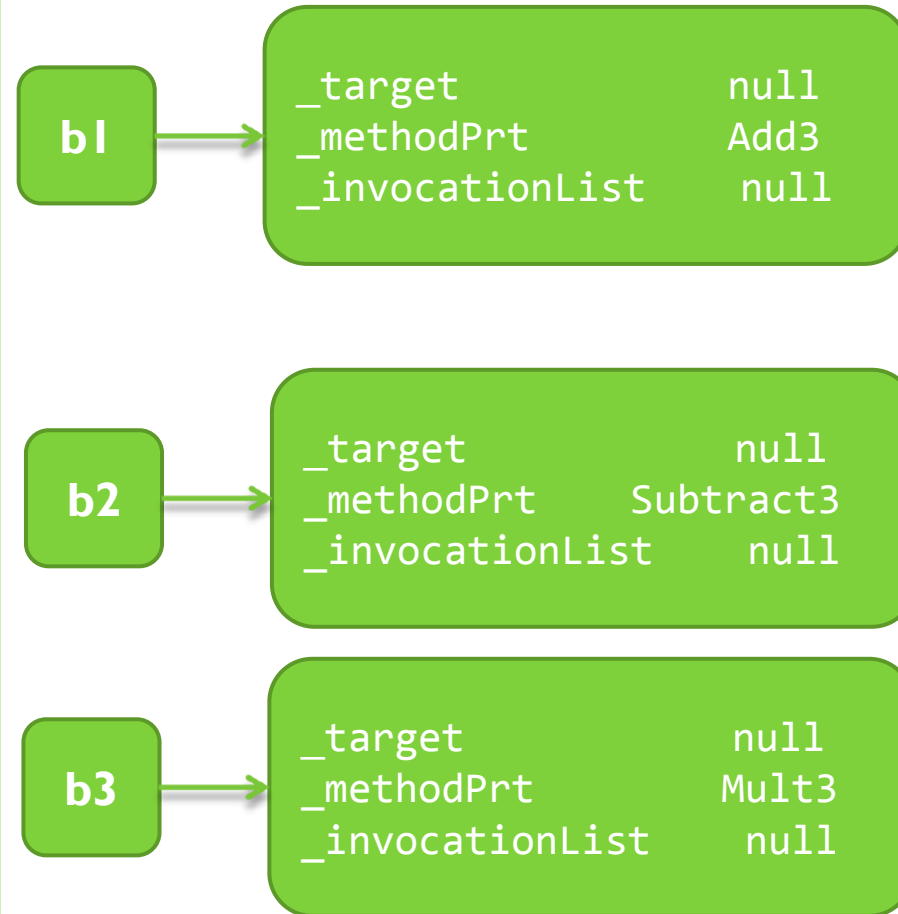
```
//...PSEUDO-CÓDIGO
using System;

public delegate void BinaryOp3(int x, int y);
public class SimpleMath{
    public void Add3 (int x, int y) {... }
    public void Subtract3 (int x, int y) {... }
    public void Mult3 (int x, int y) {... }

    //...
}

class Program{
    static void Main(string[] args){
        SimpleMath s=new SimpleMath();
        BinaryOp3 b1 = s.Add3;
        BinaryOp3 b2 = s.Subtract3;
        BinaryOp3 b3 = s.Mult3;

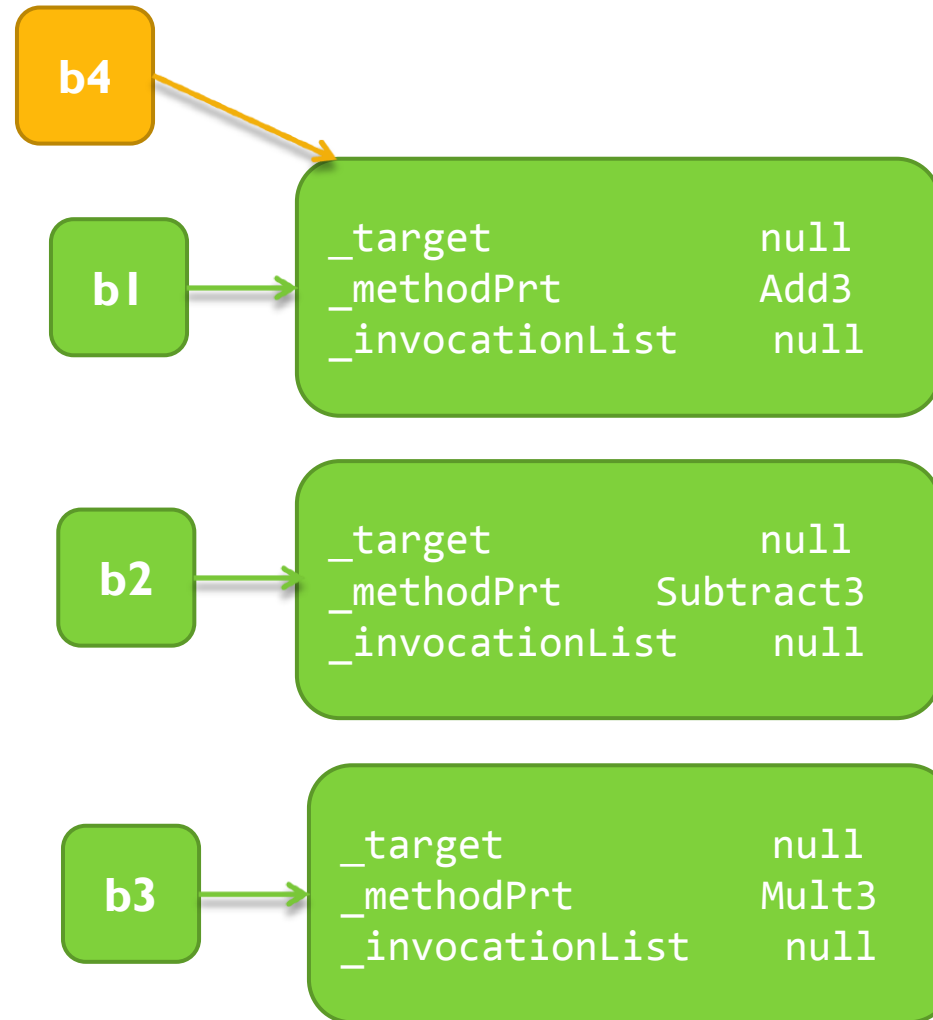
        //...
    }
}
```



Exemplo 3 (continuação)

```
//...
```

```
class Program{  
    static void Main(string[] args){  
        SimpleMath s=new SimpleMath();  
        BinaryOp3 b1 = s.Add3;  
        BinaryOp3 b2 = s.Subtract3;  
        BinaryOp3 b3 = s.Mult3;  
        BinaryOp3 b4 += b1 ;  
    }  
}
```



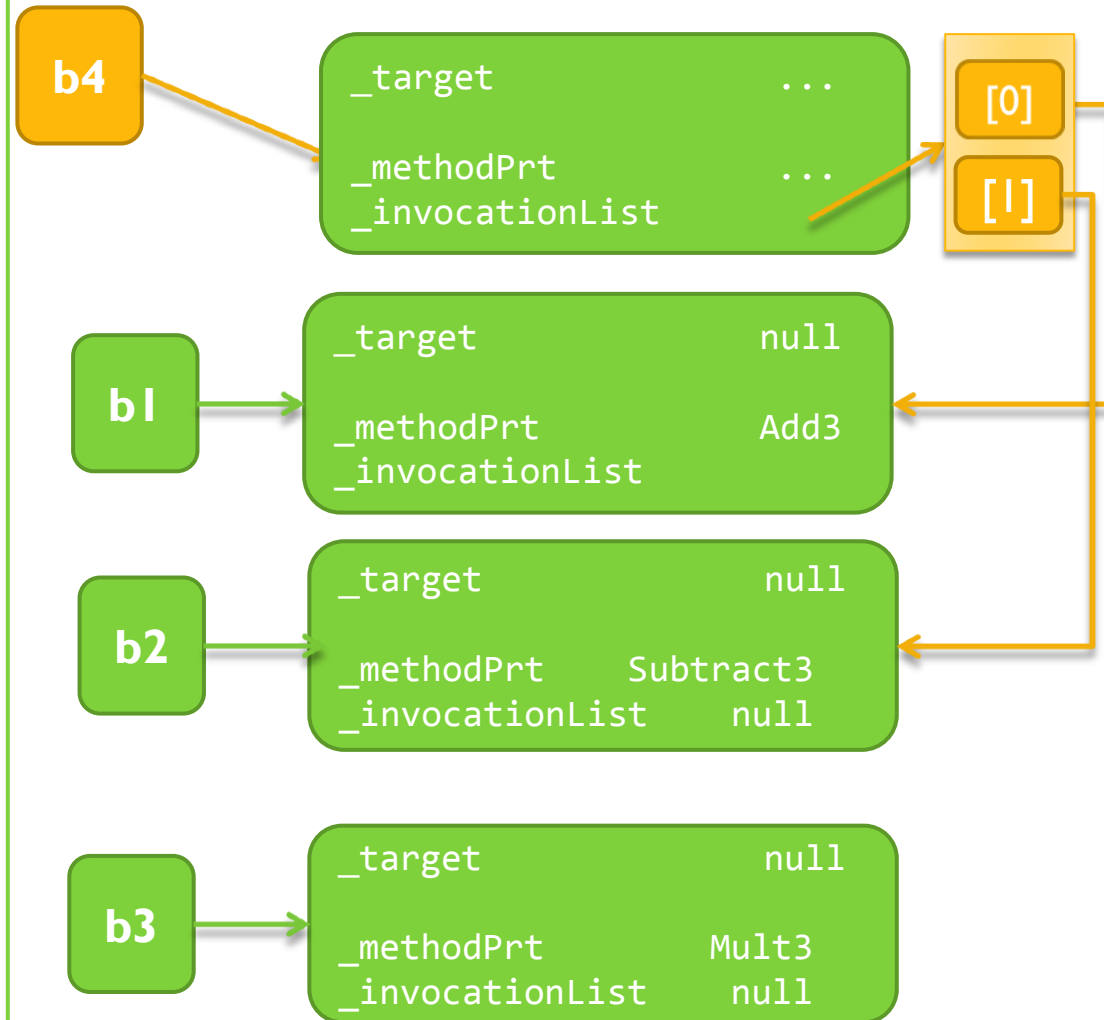
Exemplo 3 (continuação)

//...

```
class Program{  
    static void Main(string[] args){  
        SimpleMath s=new SimpleMath();  
        BinaryOp3 b1 = s.Add3;  
        BinaryOp3 b2 = s.Subtract3;  
        BinaryOp3 b4 += b1 ;  
        b4 += b2;  
    }  
}
```

//...

}}



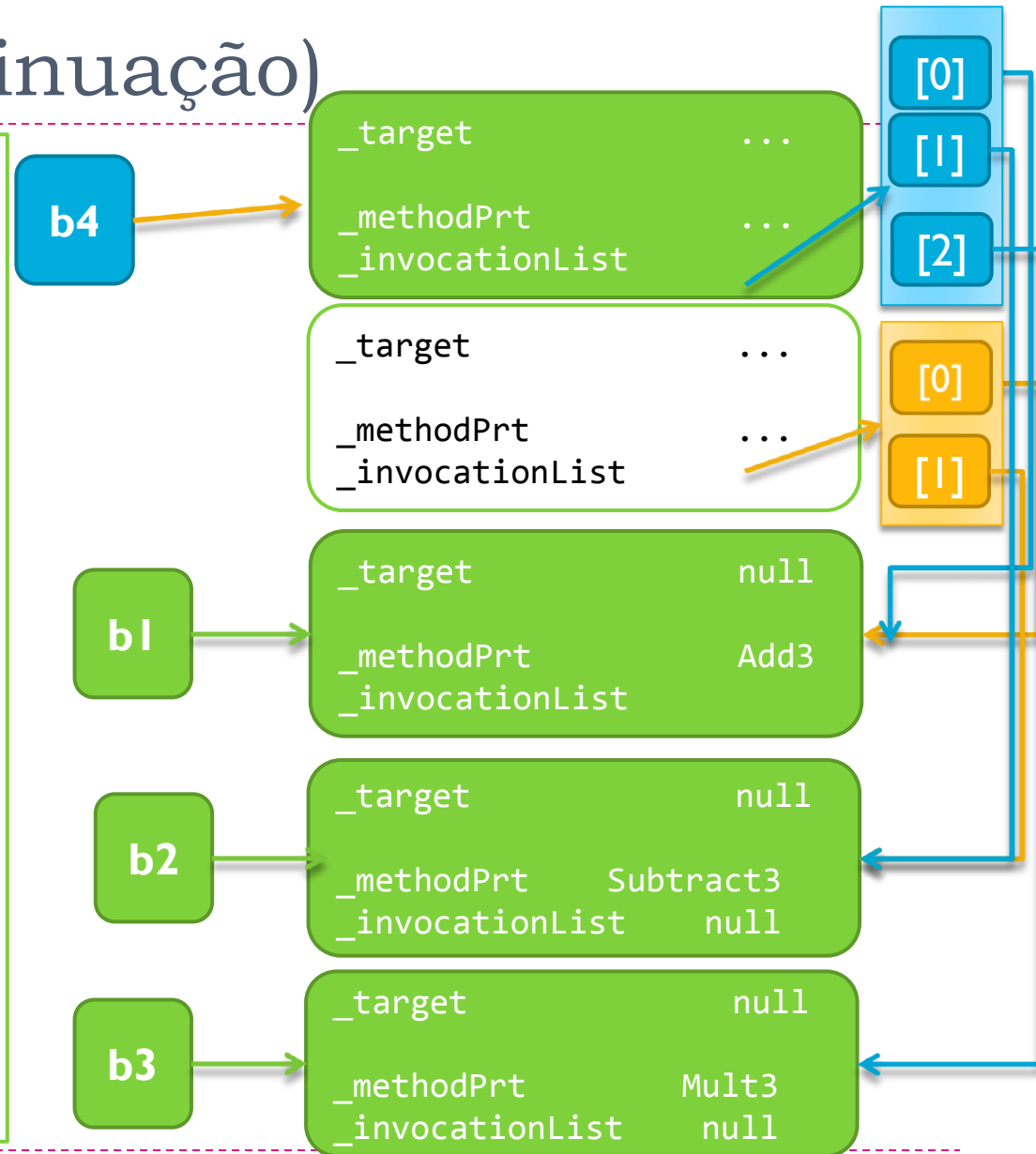
Exemplo 3 (continuação)

//...

```
class Program{  
    static void Main(string[] args){  
        SimpleMath s=new SimpleMath();  
        BinaryOp3 b1 = s.Add3;  
        BinaryOp3 b2 = s.Subtract3;  
        BinaryOp3 b4 += b1 ;  
        b4 += b2;  
        b4 += b3;  
    }  
}
```

//...

}}



Loggers – Exemplo4

```
delegate void Logger(string msg, int code);

class MyStreamLogger {
    // callback em método de instância
    StreamWriter logStream;

    public MyStreamLogger(Stream logStream) {
        this.logStream = new StreamWriter(logStream);
    }

    public void log(string s, int id) {
        logStream.WriteLine("MyStreamLogger: MSG={0}, ID={1}", s, id);
        logStream.Flush();
    }
}

class MyConsoleLogger {
    // callback em método estático
    public static void log(string s, int id) {
        System.Console.WriteLine("MyConsoleLogger:
                                MSG={0}, ID={1}", s, id);
    }
}
```

Using Loggers – Exemplo 4

```
class WorkerProcess {
    public Logger logger;

    virtual protected void doLog(string msg, int code) {
        if (logger != null)    logger(msg, code);
    }

    public void doWork() {
        // .... working....
        // logging error
        doLog("erro", 123);
    }
}

class Class1 {
    static void Main(string[] args) {
        WorkerProcess p = new WorkerProcess();
        MyStreamLogger fl = new MyStreamLogger(Console.OpenStandardOutput());
        // registrar callbacks
        p.logger += new Logger(fl.log);
        p.logger += new Logger(MyConsoleLogger.log);
        p.doWork();
    }
}
```


Métodos Anónimos

► Sabendo que:

- `public List<T> FindAll(Predicate<T> match)` é um método da classe `System.Collections.Generic.List<T>`;
- `public delegate bool Predicate<T>(T obj)` é um delegate genérico do tipo `System.Predicate<T>`;

```
//...
public class Program{
    public static void Main(){
        List<int> list = new List<int>();
        list.AddRange(new int[] {20,1,4, 8, 9});
        Predicate<int> pred;
        pred = new Predicate<int>(IsEvenNumber);
        List<int> evenNumb=list.FindAll(pred);
        //...
    }
    static bool IsEvenNumber(int i){
        return (i % 2) == 0;}
}
```

```
//...
public class Program{
    public static void Main(){
        List<int> list = new List<int>();
        list.AddRange(new int[] {20,1,4, 8, 9});
        List<int> evenNumb;
        evenNumb= list.FindAll(delegate(int i){
            return (i % 2) == 0;});
        //...
    }
}
```

Variáveis capturadas

- ▶ Variáveis externas: **variáveis locais, parâmetros valor e arrays de parâmetros** cujo scope inclua o método anónimo.
- ▶ Se o método anónimo estiver definido dentro dum método instância, então **this** também é uma variável externa
- ▶ As variáveis externas referidas pelo método anónimo dizem-se **capturadas**
- ▶ O compilador de C# cria uma classe com:
 - ▶ Um campo por cada variável capturada;
 - ▶ um método, correspondente ao método anónimo.

Exemplo

```
//...  
int n = 0;  
Del d = delegate() {  
    System.Console.WriteLine("Copy #{0}", ++n);  
};  
//...
```

- ▶ Ao contrário de variáveis locais, o tempo de vida de uma variável externa dura até os delegates que referenciam os métodos anónimos estejam elegíveis para garbage collection.
- ▶ Uma referência a `n` é capturada no momento da criação do delegate.

Variáveis capturadas (cont)

- ▶ A instanciação de um método anónimo consiste na criação de uma instância da classe referida acima e na captura do contexto.
- ▶ No entanto, existem algumas limitações:
 - ▶ Um método anónimo não pode aceder a parâmetros ref e out de um scope externo.
 - ▶ Código não seguro não pode ser acedido num bloco de um método anónimo.

Expressões Lambda

```
//...
public class Program{
    public static void Main(){
        List<int> list = new List<int>();
        list.AddRange(new int[] {20,1,4, 8, 9});
        List<int> evenNumb;
        evenNumb= list.FindAll(
            delegate(int i){
                return (i % 2) == 0;});
        //...
    }
}
```

```
//...
public class Program{
    public static void Main(){
        List<int> list = new List<int>();
        list.AddRange(new int[] {20,1,4, 8, 9});
        List<int> evenNumb;
        evenNumb= list.FindAll(i => (i % 2) == 0);
        //...
    }
}
```

- ▶ O compilador de C# traduz a expressão lambda para um método anónimo, usando o tipo delegate Predicate<T>

Expressões Lambda com múltiplas instruções

```
//...  
public class Program{  
    public static void Main(){  
        List<int> list = new List<int>();  
        list.AddRange(new int[] {20,1,4, 8, 9});  
        List<int> evenNumb;  
        evenNumbers = list.FindAll((i) =>  
            {  
                Console.WriteLine("value of i is currently: {0}", i);  
                bool isEven = ((i % 2) == 0);  
                return isEven;  
            });  
        //...  
    }  
}
```

Expressões Lambda com múltiplos argumentos

```
//...
public class SimpleMath{
    public delegate void
        MathMessage(string msg, int result);
    private MathMessage mmDelegate;
    public void
        SetMathHandler(MathMessage target){
            mmDelegate = target; }

    public void Add(int x, int y){
        if (mmDelegate != null)
            mmDelegate.Invoke("Adding has
                               completed!", x + y);
    }
}
```

```
//...
static void Main(string[] args){
    SimpleMath m = new SimpleMath();
    m.SetMathHandler(
        (msg, result) =>
            {Console.WriteLine("Message: {0},
                               Result: {1}", msg, result);}
    );
    // Executa a expressão Lambda
    m.Add(10, 10);
}
```

Enumeráveis e enumeradores

- ▶ Quando um tipo é passível de ser enumerado, deve implementar a interface **IEnumerable<T>**, que contém o único método:
 - ▶ `IEnumerator<T> GetEnumerator()`
- ▶ Admitindo que **en** é enumerável, a construção
 - ▶ `foreach(T t in en){ body }` é traduzida em:
 - ▶

```
IEnumerator<T> enumerator1=en.GetEnumerator()) {  
    while (enumerator1.MoveNext()) {  
        T t =enumerator1.Current;  
        //body }  
}
```


Interfaces genéricas e não genéricas

- ▶ **IEnumerable<T> : IEnumerable**
 - ▶ Método não genérico **IEnumerator GetEnumerator()**, com implementação de forma explícita
 - ▶ Método genérico **IEnumerator<T> GetEnumerator()**
- ▶ **IEnumerator<T> : IDisposable, IEnumerator**
 - ▶ Acrescenta a interface **IDisposable**
 - ▶ Métodos **Reset** e **MoveNext** são de **IEnumerator**
 - ▶ Duas propriedades **Current**
 - ▶ Genérica, retorna **T**
 - ▶ Não genérica, retorna **object**—implementada de forma explícita

Utilização de IEnumerable/IEnumerator

- ▶ A utilização de enumeradores sobre tipos enumeráveis pode ter dois tipos de implementações/utilizações:
 - ▶ Sequências onde os elementos já estão calculados e armazenados numa estrutura de dados **(Solução 1)**
 - ▶ Sequências onde os elementos são calculados apenas quando necessários –aquando da chamada do método MoveNext **(Solução 2)**

Exemplo de um Problema - Filtro

Dada uma sequência **seq** e um predicado **pred**, obter a sequência **newSeq** com os elementos de **seq** que satisfazem **pred**

```
IEnumerable<T> Filter<T> (IEnumerable<T> seq, Predicate<T> pred)
```

Solução 1 (eager)

- ▶ Sequências onde os elementos já estão calculados e armazenados numa estrutura de dados

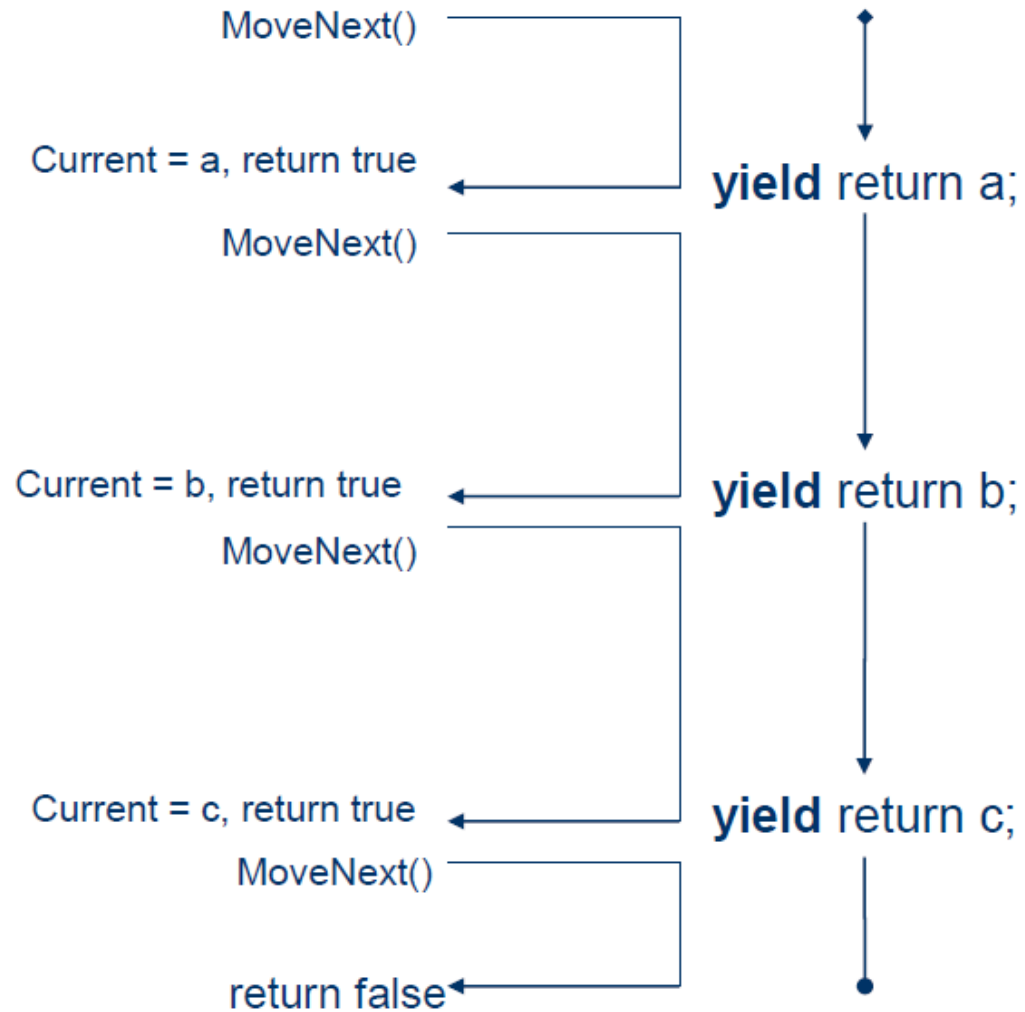
```
public static IEnumerable<T> Filter<T> (IEnumerable<T> seq,  
                                         Predicate<T> pred){  
    List<T> result= new List<T>();  
    foreach(T t in seq){  
        if(pred(t)) result.Add(t);  
    }  
    return result;  
}
```

Solução 2 (lasy)

- ▶ Sequências onde os elementos são calculados apenas quando necessários

```
public static IEnumerable<T>
    Filter<T>(IEnumerable<T> seq, Predicate<T> pred){
        foreach(T t in seq){
            if( pred(t) )
                yield return t; // result.Add(t);
        }
    }
```

yield



Exemplos

```
for (int i = 0; i < 5; i++) {  
    yield return i;  
}  
Console.Out.WriteLine("Aparece");
```

```
int i = 0;  
while (true) {  
    if (i < 5) {  
        yield return i;  
    }  
    else {  
        // note que i++ não vai ser executado depois disto  
        yield break;  
    }  
    i++;  
}  
Console.Out.WriteLine("Não aparece");
```

Iteradores

```
public static IEnumerable<T>
    Filter<T>(IEnumerable<T> ie, Predicate<T> pred){
        foreach(T t in ie) {
            if (pred(t)) yield return t;
        }
    }
```

- ▶ O método **Filter** retorna uma classe gerada pelo compilador e que implementa **IEnumerable<T>** e **IEnumerator<T>**
 - ▶ Os seus métodos, nomeadamente o **MoveNext**, reflectem a sequência de acções definida no corpo da função geradora
 - ▶ O *contexto da geração é capturado* para ser usado no método **MoveNext**
- ▶ Sintaxe e semântica
 - ▶ **yield return t**
 - ▶ sinaliza que o fio de execução (do **MoveNext**) termina com **true** e **Current = t**
 - ▶ **yield break**
 - ▶ sinaliza que o fio de execução (do **MoveNext**) termina com **false** (**Current** é indeterminado)

Iteradores

Classe gerada pelo compilador com base no corpo da função Filter

```
class X:
    IEnumerator<T>, IEnumerable<T>{
        T Current{ get{. . .}};
        bool MoveNext( ){
            //maquina de estados
        }
        IEnumerable<T> ie;
        Predicate<T> pred;
    }
```

returns

```
IEnumerator<T>
Filter<T>(IEnumerable<T> ie,
          Predicate<T> pred){
    foreach(T t in ie){
        if(pred(t)) yield return t;
    }
}
```

Variáveis Capturadas

- ▶ O método **MoveNext** implementado através duma máquina de estados
 - ▶ Estado -2: ainda não foi obtido o enumerador
 - ▶ Estado 0: enumerador no estado inicial
 - ▶ Estado -1: enumerador no estado final

Mais Exemplos

```
public static IEnumerable<U>
    SelectMany<T,U>(IEnumerable<T> seq, Converter<T, IEnumerable<U>> convert)
    {
        foreach(T t in seq){
            foreach(U u in convert(t)){ yield return u; }
        }
    }
```

```
public static void Main(){
    List<int> list1=new List<int>(new int[]{2,3,4});
    List<int> list2=new List<int>(new int[]{1,2,3,4,5,6,7,8,9,10});
    IEnumerable<int> en = SelectMany<int,int>(
        list1, i => list2.FindAll(j => j%i==0 ));
    foreach(int i in en)
        Console.WriteLine(i);
}
```

Mais Exemplos

```
public static IEnumerable<T>
    Append<T>(IEnumerable<T> seq1, IEnumerable<T> seq2){
    foreach(T t in seq1) yield return t;
    foreach(T t in seq2) yield return t;
}
```

Delegates e algoritmos genéricos pré-definidos

- ▶ No *namespace* `System` estão definidos 4 *delegates* genéricos:

```
public delegate void Action<T> (T obj)
public delegate int Comparison<T> (T x, T y)
public delegate TOutput Converter<TInput, TOutput> (TInput input)
public delegate bool Predicate<T> (T obj)
public delegate TResult Func<out TResult>()
```

- ▶ As classes `System.Collections.Generic.List<T>` e `System.Array` disponibilizam um conjunto de métodos, parametrizados por **functores**, para acesso aos seus dados :

List<T>:

```
public int FindIndex ( Predicate<T> match );
public List<T> FindAll ( Predicate<T> match );
public bool TrueForAll ( Predicate<T> match );
public void ForEach ( Action<T> action );
```

Array:

```
public static void ForEach<T> ( T[] array, Action<T> action );
public static T Find<T> ( T[] array, Predicate<T> match );
public static bool Exists<T> ( T[] array, Predicate<T> match );
public static void Sort<T> ( T[] array, Comparison<T> comparison );
public static U[] ConvertAll<T, U> ( T[] array, Converter<T,U> converter );
```

Exemplo: Utilização do delegate **Predicate**

```
public static int countIf(int[] a, Predicate<int> pred) {  
    int c=0;  
    foreach(int i in a)  
        if ( pred(i) ) c++;  
    return c;  
}
```

```
public static int countEven(int[] a) {  
    return countIf( a, i => i%2 == 0 );  
}
```

```
public static int countOdd(int[] a) {  
    return countIf( a, i => i%2 != 0 );  
}
```

Exemplo: Utilização do delegate **Action**

```
public static List<int> FactorList(List<int> list, int factor){  
    List<int> newList =new List<int>();  
    list.ForEach( i => {newList.Add( factor * i );} );  
    return newList;  
}
```

```
public static void Main(){  
    List<int> values=new List<int>();  
    values.AddRange(new int[]{1,2,3,4,5,6,7,8,9});  
    List<int> newValues= FactorList(values,2);  
    foreach(int j in newValues)  
        Console.WriteLine(j);  
}
```

Exemplo 1: Utilização do delegate **Converter**

```
public static int[] ModN( int[] array, int n ){  
    int[] newArray = Array.ConvertAll<int,int>(array, i => i%n );  
    return newArray;  
}
```

```
public static void Main(){  
    int[] values={1,2,3,4,5,6,7,8,9};  
    int[] newValues=ModN(values,3);  
    foreach(int j in newValues)  
        Console.WriteLine(j);  
}
```

Exemplo: Utilização do delegate **Comparison**

- ▶ O método **OrderBy** enumera de forma ordenada uma sequencia, segundo a chave produzida pela *função* **sortkey**


```
public static IEnumerable<T>
    OrderBy<T,U>(IEnumerable<T> seq, Converter<T,U> sortkey)
        where U : IComparable<U>{

    List<T> list = newList<T>( seq );
    list.Sort( (T t1, T t2) => sortkey(t1).CompareTo(sortkey(t2)) );
    return list;
}
```


Construção de Pipelines

```
class Program{  
    static public IEnumerable<T> Filter<T>(IEnumerable<T> col, Predicate<T> pred){  
        foreach(T item in col) if(pred(item)) yield return item;  
    }  
    static public IEnumerable<T> Transform<T>(IEnumerable<T> col, Converter<T,T> transform){  
        foreach(T item in col) yield return transform(item);  
    }  
    static public IEnumerable<U> Converter<T,U>(IEnumerable<T> col, Converter<T,U> convert){  
        foreach(T item in col) yield return convert(item);  
    }  
    static public IEnumerable<T> Printer<T>(IEnumerable<T> col){  
        foreach(T item in col) Console.WriteLine(item);  
        return collection;  
    }  
}//...  
}
```

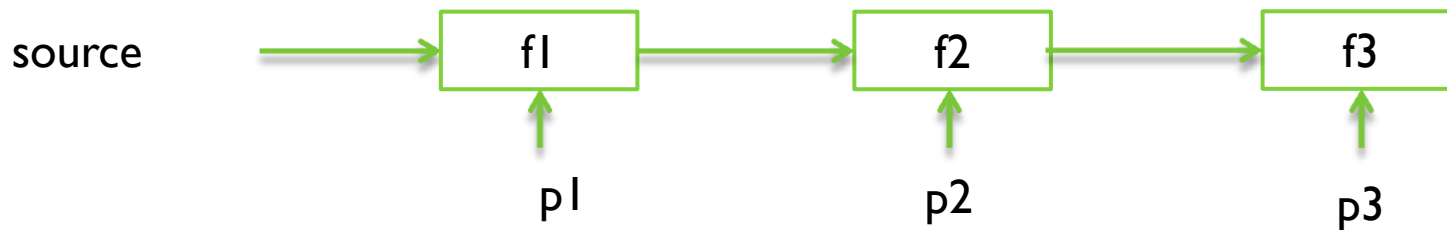
```
public static void Main(){  
    List<int> list=new List<int>(new int[]{1,2,3,4,5,6,7,8,9,10});  
    Printer(  
        Converter<int,string>(  
            Filter(Transform(list, item => item*2), item => item%3==0),  
            item => "Hello"+item.ToString() )  
    );  
}}
```



Hello6
Hello12
Hello18

Métodos de Extensão


- ▶ Métodos estáticos invocáveis usando a sintaxe de método de instância
- ▶ Utilização: simplificar a sintaxe de construção de *pipelines*
- ▶ Com métodos estáticos
 - ▶ `f3(f2(f1(source,p1),p2),p3)`
- ▶ Com métodos instância
 - ▶ `source.f1(p1).f2(p2).f3(p3)`
- ▶ Métodos extensão (têm de estar presentes em classes estáticas)
 - ▶ `IEnumerable<T> f1(this IEnumerable<T>, P1 p1)`



Construção de Pipelines

```
class Program{
    static public IEnumerable<T> Filter<T>(this IEnumerable<T> col, Predicate<T> pred){
        foreach(T item in col) if(pred(item)) yield return item;
    }
    static public IEnumerable<T> Transform<T>(this IEnumerable<T> col, Converter<T,T> transform){
        foreach(T item in col) yield return transform(item);
    }
    static public IEnumerable<U> Converter<T,U>(this IEnumerable<T> col, Converter<T,U> convert){
        foreach(T item in col) yield return convert(item);
    }
    static public IEnumerable<T> Printer<T>(this IEnumerable<T> col){
        foreach(T item in col) Console.WriteLine(item);
        return collection;
    }//...
}
```

```
public static void Main(){
    List<int> list=new List<int>(new int[]{1,2,3,4,5,6,7,8,9,10});
    list.Transform(item => item*2).Filter(item=>item%3==0).
    Converter<int,String>(item =>"Hello"+item.ToString()).Printer();
}}
```



Hello6
Hello12
Hello18

System.Linq

- ▶ **System.Linq.Enumerable** define um conjunto de métodos de extensão sobre **IEnumerable<T>**
- ▶ Exemplos
 - ▶ Restrição: **Where**
 - ▶ Projecção: **Select, SelectMany**
 - ▶ Ordenação: **OrderBy, ThenBy**
 - ▶ Agrupamento: **GroupBy**
 - ▶ Quantificadores: **Any, All**
 - ▶ Partição: **Take, Skip, TakeWhile, SkipWhile**
 - ▶ Conjuntos: **Distinct, Union, Intersect, Except**
 - ▶ Elementos: **First, FirstOrDefault, ElementAt**
 - ▶ Agregação: **Count, Sum, Min, Max, Average**
 - ▶ Conversão: **ToArray, ToList, ToDictionary**

Exemplo

```
static public
    IEnumerable<T> Filter<T>(this IEnumerable<T> col, Predicate<T> pred){
        foreach(T item in collection)
            if(pred(item)) yield return item;
    }
```

```
static public
    IEnumerable<T> Filter<T>(this IEnumerable<T> col, Predicate<T> pred){
        return collection.Where(i=>predicate(i));
    }
```

Tipificação implícita de variáveis locais

- ▶ Inferência do tipo das variáveis locais com base no tipo da sua iniciação
- ▶ Exemplos
 - ▶ `var i = 5;`
 - ▶ `var s = new string("aaa");`
 - ▶ `var s = "aaa";`
 - ▶ `var point = {X = 3, Y = 4}`
- ▶ Não é tipificação dinâmica, é tipificação estática com inferência do tipo em tempo de compilação
- ▶ Utilização
 - ▶ Tipos complexos
 - ▶ Tipos anónimos
- ▶ Só pode ser usada em variáveis locais
 - ▶ Não pode ser usada no tipo de retorno ou no tipo dos parâmetros

Tipos anónimos

- ▶ Criação automática de tipos para o armazenamento de tuplos
 - ▶ `var point = new { X = 3, Y = 4 };`
 - ▶ Resulta na criação duma classe anónima com as propriedades públicas `X` e `Y`, do tipo `int`, inferidas do iniciador `{X=3, Y=4}`
- ▶ Na mesma unidade de compilação, dois iniciadores iguais utilizam o mesmo tipo anónimo
- ▶ Forma alternativa
 - ▶ `int X = 3;`
 - ▶ `int Y = 4`
 - ▶ `var point = new {X, Y}`
- ▶ Tipo anónimo associado tem as propriedades **X** e **Y**

Exemplo

- ▶ Para cada ficheiro com extensão “.cs” apresentar o nome do ficheiro e o respectivo número de linhas

```
public static void ShowNumberOfLines(string path) {  
    DirectoryInfo directory = new DirectoryInfo(path);  
    var files = directory.EnumerateFiles().  
        Where(fi => fi.Extension.Equals(".cs")).  
        Select(fi => new { X = fi, Lines = countLines(fi) });  
    foreach (var p in files) {  
        Console.WriteLine("file = {0}; lines = {1}", p.X.FullName, p.Lines);  
    }  
}
```

```
public static int countLines(FileInfo f){  
    int count=0;  
    StreamReader sr = new StreamReader(f.OpenRead());  
    while (sr.Peek() != -1){  
        sr.ReadLine();  
        count++;  
    }  
    return count;  
}
```


O mesmo exemplo tirando partido da sintaxe Linq

- ▶ Para cada ficheiro com extensão “.cs” apresentar o nome do ficheiro e o respectivo número de linhas

```
public static void ShowNumberOfLines(string path) {  
    DirectoryInfo directory = new DirectoryInfo(path);  
    var files = from fi in directory.EnumerateFiles()  
                where fi.Extension.Equals(".cs")  
                select new { X = fi, Lines = countLines(fi)};  
    foreach (var p in files) {  
        Console.WriteLine("file = {0}; lines ={1}", p.X.FullName, p.Lines);  
    }  
}
```