ISEL

Ambientes Virtuais de Execução

2021

Week 10 – Sequences

higher-order functions

Ate the end of this Chapter

- Understand Higher-order functions

e.g. …`filter(item => …).map(item =>….).forEach(…)`

Streams allow *pipeline composition*

- https://martinfowler.com/articles/collection-pipeline/
- https://developer.ibm.com/articles/j-java-streams-3-brian-goetz

Pipeline:

- Operations chaining
- The result of an *operation* is the parameter for the next *operation*

```
Chained:      …filter(item => …).map(item =>….).forEach(…)
```

The result of filter is the this parameter of the map.

```
Nested:     forEach(map(...filter()))
```

Sequences/Streams

- []
- stream (ALGOL 1965)
- list (LISP 1976)
- Iterator (C++ STL 1994)
- Iterator (Java 1.2 1998)
- IEnumerable (.net 2002)
- Stream (Java 8 2014)
- Reactive Streams (Java 9, RxJava 3, Reactor)
- Async Iterator (ES2018 and C#8 2019)
- Kotlin Flow (2019)

- Collection ~ in memory
- Sequence may not be a Collection

- Any programming language provides an abstraction for Sequences.
- Even to day we may find innovation on Sequences

E.g. isel-AVE-2021.txt *data source*

Sequence of Strings

```
static IEnumerable Lines(string path)
{
    string line;
    IList res = new ArrayList();
    using(StreamReader file = new StreamReader(path))
    {
        while ((line = file.ReadLine()) != null)
        {
            res.Add(line);
        }
    }
    return res;
}
```

Exercise

Objectivo:

1. Listar o 1º nome dos alunos – `ConvertToFirstName`
2. Que começem com a letra “D” – `FilterNameStartsWith(…, prefix)`
3. Que tenham o número maior que 47000
   - `FilterWithNumberGreaterThan(…, nr)`
4. Converter String em Student - `ConvertToStudent()`

Exercise… e.g. nested form

```
IEnumerable names =
    ConvertToFirstName(                    // Seq<String>
        FilterNameStartsWith(              // Seq<Student>
            FilterWithNumberGreaterThan( // Seq<Student>
                ConvertToStudents(         // Seq<Student>
                    Lines("isel-AVE-2021.txt")),  // Seq<String>
                47000),
            "D")
        );
```

We read in inverse order of execution!

FilterWithNumberGreaterThan…

```
IEnumerable FilterWithNumberGreaterThan(IEnumerable stds, int nr)
```

- Version 1 naif

```
    static IEnumerable FilterWithNumberGreaterThan(IEnumerable stds, int nr) {
        IList res = new ArrayList();
        foreach (object o in stds) {
            if (((Student)o).Number > nr) res.Add(o);
        }
        return res;
    }
```

Homework: Add a Distinct operation to the pipeline

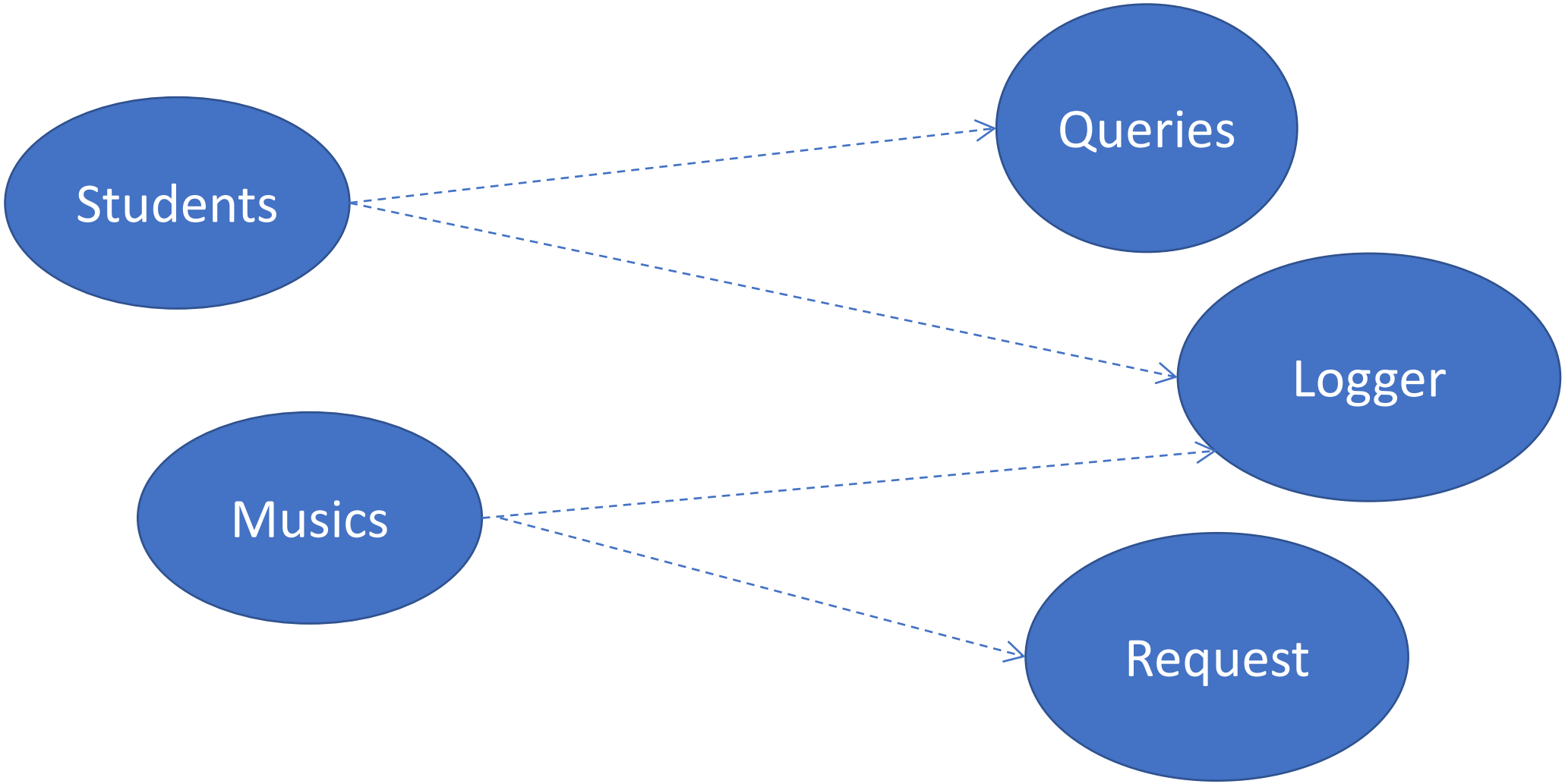
```
IEnumerable names =
            Distinct(
                ConvertToFirstName(                    // Seq<String>
                    FilterNameStartsWith(              // Seq<Student>
                        FilterWithNumberGreaterThan( // Seq<Student>
                            ConvertToStudents(         // Seq<Student>
                                Lines("isel-AVE-2021.txt")),  // Seq<String>
                            47000),
                        "D")
                    )
                );
```

Versio 1 – naif

Problems:

- Repeated code (i.e. `foreach (object o in stds) { Student std = (Student)o;...`)
- Cannot be reused in different **Domain !!!!!**

Domain versus Util



Domínio versus Util

- Fonte de dados (e.g. BD, Web API Retsful, ficheiro i41d.txt)

- Dominio e.g. Students:
  - Pesquisa de alunos com numero maior que 45000
  - Alunos com o nome a iniciar por J
  - Etc…
  - !!!!!! Não queremos fazer processamento com String….

- Util – operações para:
  - Fltrar
  - Transformar
  - etc

Remove Domain specific form Convert

```
IList res = new ArrayList();
foreach (object o in src) {
    res.Add(Student.Parse((string) o));
}
return res;
```

```
IList res = new ArrayList();
foreach (object o in src) {
    res.Add(((Student)o).Name.Split(" ")[0]);
}
return res;
```

Behavior parametrization

```
static IEnumerable Convert(IEnumerable src, ...) {
```

```
public interface Function {
    object Invoke(object o);
}
```

Remove Domain specific form Filter

```
IList res = new ArrayList();
foreach (object o in src) {
    if ((Student)o().Number > nr)
      res.Add(o);
}
return res;
```

```
IList res = new ArrayList();
foreach (object o in src) {
    if (((Student)o).Name.StartsWith(prefix))
      res.Add(o);
}
return res;
```

Behavior parametrization

```
static IEnumerable Filter(IEnumerable stds, ...) {
```

```
public interface Predicate {
    ...Invoke(…. o);
}
```

Queries 3 - delegates

```
IEnumerable names =
        Convert(                // Seq<String>
            Filter(             // Seq<Student>
                Filter(         // Seq<Student>
                    Convert(    // Seq<Student>
                        Lines("isel-AVE-2021.txt"),
                        new ToStudent()),
                        ...
```

Object Oriented approach

```
class ToStudent : Function
{
    public object Invoke(object o)
    {
        return Student.Parse((string) o);
    }
}
```

1. ***Define a class implementing an interface***
2. ***Instantiate that class***

```
IEnumerable names =
        Convert(                // Seq<String>
            Filter(             // Seq<Student>
                Filter(         // Seq<Student>
                    Convert(    // Seq<Student>
                        Lines("isel-AVE-2021.txt"),
                        AppQueries3.ToStudent),
                        ...
```

Functional approach

```
private static object ToStudent(object o)
{
    return Student.Parse((string) o);
}
```

1. ***Define a method compatible with*** `FunctionDelegate`
2. ***Pass a method reference (i.e. method handle)***

Be careful

`AppQueries3.ToStudent != AppQueries3.ToStudent()`

method reference *!=* *method call*

E.g. Java

`AppQueries3::ToStudent != AppQueries3.ToStudent()`

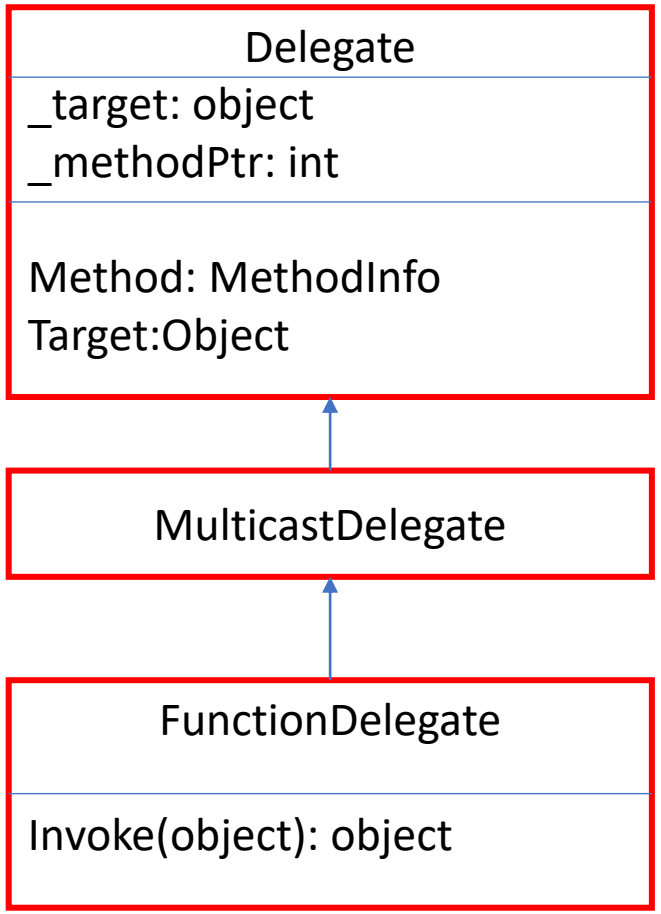
Method Reference => Delegate instance

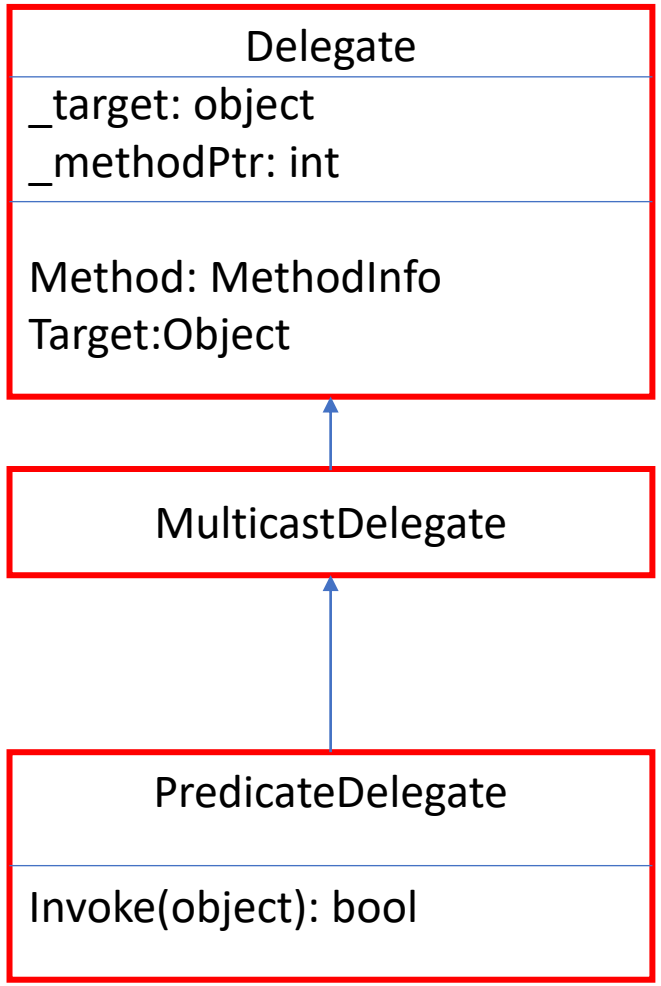
```
Convert(
    Lines("isel-AVE-2021.txt"),
    AppQueries3.ToStudent),
...
```

```
Convert(
    Lines("isel-AVE-2021.txt"),
    new FunctionDelegate(AppQueries3.ToStudent)),
```

```
IEnumerable Convert(IEnumerable src, FunctionDelegate mapper)
```

Delegate internals





E.g. AppQueries3.ToStudent)

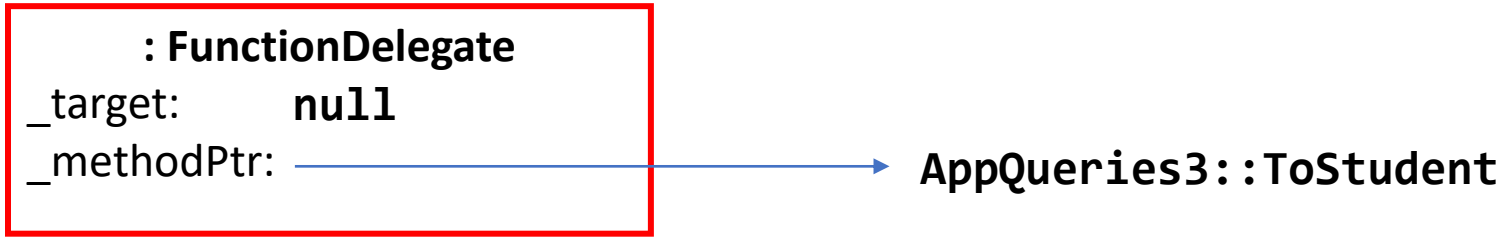
```
Convert(
    Lines("isel-AVE-2021.txt"),
    AppQueries3.ToStudent),
...
```

```
Convert(
    Lines("isel-AVE-2021.txt"),
    new FunctionDelegate(AppQueries3.ToStudent)),
```

: FunctionDelegate
_target: **null**
_methodPtr: → **AppQueries3::ToStudent**

E.g. AppQueries3.ToStudent)

```
ldnull
ldftn      object AppQueries3::ToFirstName(object)
newobj     instance void FunctionDelegate::.ctor(object, native int)
```



Lambda Expression …. => … or … -> … {…->… }

- Anonymous method
  (“without name” => The name is generated by the compiler)
- Concise (or compact) definition of a method
- E.g.

```
private static object ToStudent(object o)
{
    return Student.Parse((string) o);
}
```

```
o => Student.Parse((string) o)
```

parameters

Method Body

The return is implicit

Equivalent forms

```
o => Student.Parse((string) o)
```

```
(object o) => Student.Parse((string) o)
```

```
(object o) => { return Student.Parse((string) o); }
```

Lambda Expression…

```
.method object  '<Main>b__3_0'(object o) cil managed
{
  IL_0000:  ldarg.1
  IL_0001:  castclass  [System.Runtime]System.String
  IL_0006:  call       class Student Student::Parse(string)
  IL_000b:  ret
} // end of method '<>c'::'<Main>b__3_0'
```

```
o => Student.Parse((string) o)
```

```
ldsfld     class AppQueries3/'<>c' AppQueries3/'<>c'::'<>9'
ldftn      instance object AppQueries3/'<>c'::'<Main>b__3_0'(object)
newobj     instance void FunctionDelegate::.ctor(object, native int)
```

: FunctionDelegate

_target: → **: AppQueries3/'<>c'**

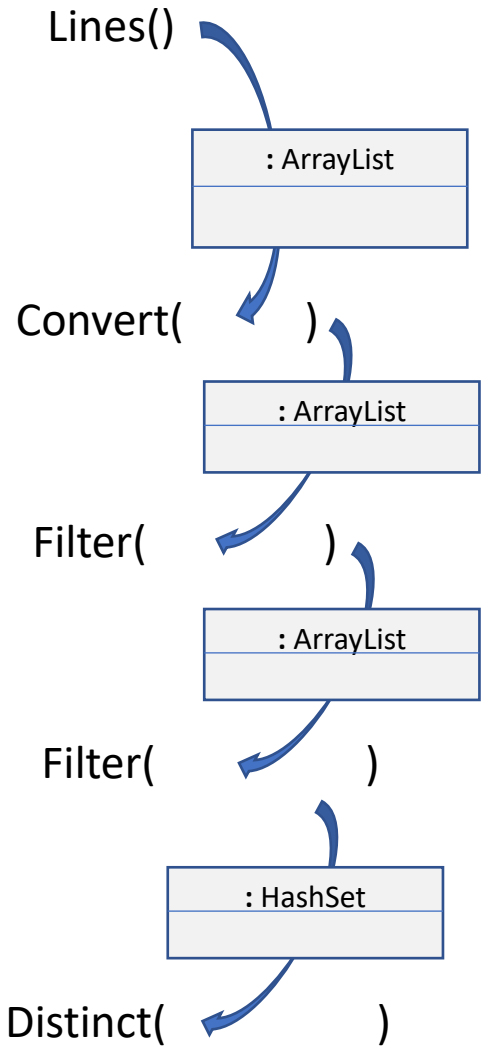
_methodPtr: → **AppQueries3/'<>c'::'<Main>b__3_0'(object)**

Versão 4 - Eager

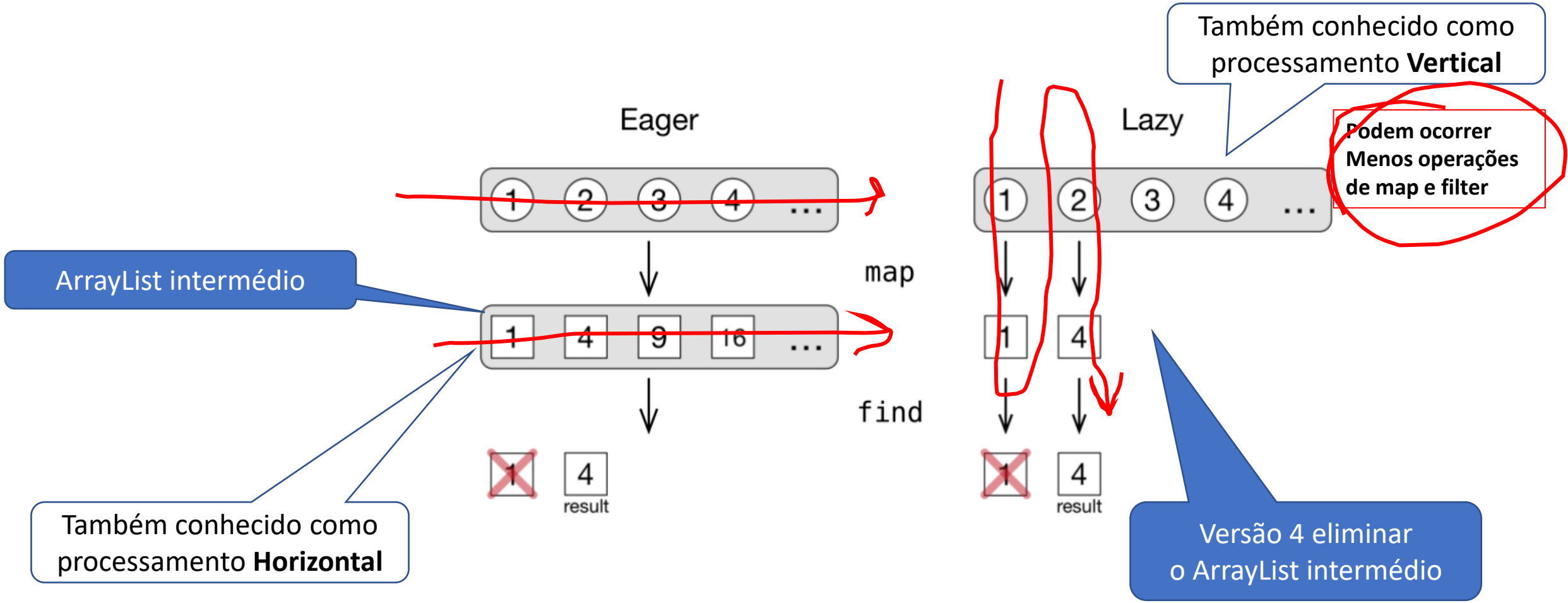
- Versão 4 overheads: Colecções intermédias
- Num **pipeline** de operações as colecção intermédias são limpas pelo **GC**
- Resultado de uma operação => parametro da operação seguinte

(*) *collection pipeline, cite Martin Fowler*

```
IEnumerable names =
            Convert (
                Filter (
                    Filter (
                        Convert (
                            Lines("i41d.txt")),
                        …),
                    …)
                …);
```



Versão 4- Eager<versus> Versão 5- Lazy



Versão 5- Lazy

Observação do comportamento lazy:

- Até que seja executada a **operação terminal (e.g. foreach)** nenhuma lambda passada por parametro é executada.
- **Intercalação** entre operações intermédias. Cada elemento atravessa todas as operações intermédias antes de processor o próximo elemento.

- Version 5 overheads: **Verbose** implementation of Iterator. Example:

```
class EnumerableFilter : IEnumerable, IEnumerator {
    ...
    public EnumerableFilter(IEnumerable src,  Predicate pred) {
        this.src = src;
        this.pred = pred;
    }
    public IEnumerator GetEnumerator() {
        this.iter = src.GetEnumerator();
        return this;
    }
    public bool MoveNext() {
        while(iter.MoveNext()) {
            current = iter.Current;
            if(pred.Invoke(current))
                return true;
        }
        current = null;
        return false;
    }
    public object Current { get { return current; }}
    public void Reset() {
        current = null;
        iter.Reset();
    }
}
```

Version 5: Lazy but Verbose. Hard to read.

Version 3: Concise but Eager

```
IEnumerable Filter(IEnumerable src, Predicate pred) {
    IList res = new ArrayList();
    IEnumerator iter = src.GetEnumerator();
    while (iter.MoveNext())
    {
        if(pred.Invoke(iter.Current))
            res.Add(iter.Current);
    }
    return res;
}
```

Version 6 goal: Concise and Lazy

…

Version 6: generators

Generators:

- are **functions/methods**.
- may yield **multiple values** instead of returning one value a single time.
- are **lazily** computed.
- calling a generator function **does not** execute its body immediately.
- the execution of the generator's body is resumed after the *yield* action.

https://en.wikipedia.org/wiki/Generator_(computer_programming)

Versão 6 – Lazy yield

```
static IEnumerable Convert() {
    IList res = new ArrayList();
    foreach (object o in src) {
        res.Add(mapper(o));
    }
    return res;
}
```

```
static IEnumerable Convert(...) {

    foreach (object o in src) {
        yield return mapper(o);
    }

}
```

Version 6: generators

Generators:

- are **functions/methods**.
- may yield **multiple values** instead of returning one value a single time.
- are **lazily** computed.
- calling a generator function **does not** execute its body immediately.
- the execution of the generator's body is **resumed after the last *yield* action**.

```
static IEnumerable Numbers() {

  Console.WriteLine("Iteration started...");

  yield return 11;

  yield return 17;

  yield return 23;
}
```

Version 6: generators

Generators:

- are **functions/methods**.
- may yield **multiple values** instead of returning one value a single time.
- are **lazily** computed.
- calling a generator function **does not** execute its body immediately.
- the execution of the generator's body is **resumed after the last *yield* action**.

```
static IEnumerable Numbers() {

  Console.WriteLine("Iteration started...");

  yield return 11;

  yield return 17;

  yield return 23;
}
```

```
IEnumerator nrs = Numbers()
                    .GetEnumerator();

Console.ReadLine();

nrs.MoveNext();

Console.WriteLine(nrs.Current);

nrs.MoveNext();

Console.WriteLine(nrs.Current);

nrs.MoveNext();

Console.WriteLine(nrs.Current);
```

Version 6: generators

Version 2: Concise but Eager

```
IEnumerable Filter(IEnumerable src, Predicate pred) {
   IList res = new ArrayList();
   IEnumerator iter = src.GetEnumerator();
   while (iter.MoveNext())
   {
       if(pred.Invoke(iter.Current))
          res.Add(iter.Current);
   }
   return res;
}
```

Version 5: Concise and Lazy

```
IEnumerable Filter(IEnumerable src, Predicate pred) {
   IList res = new ArrayList();
   IEnumerator iter = src.GetEnumerator();
   while (iter.MoveNext())
   {
       if(pred.Invoke(iter.Current))
          res.Add( yield return iter.Current );
   }
   return res;
}
```