



Curso de Extensão  
Tecnologias Microsoft

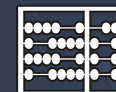


INF0992

# Programação Avançada em C#

Hervé Yviquel  
hyviquel@unicamp.br

24 de Setembro de 2022



INSTITUTO DE  
COMPUTAÇÃO

The background features a network of gray lines connecting various colored circles (orange, yellow, blue, green) and a dark blue horizontal band with a white circuit-like pattern.

# Apresentação do Módulo



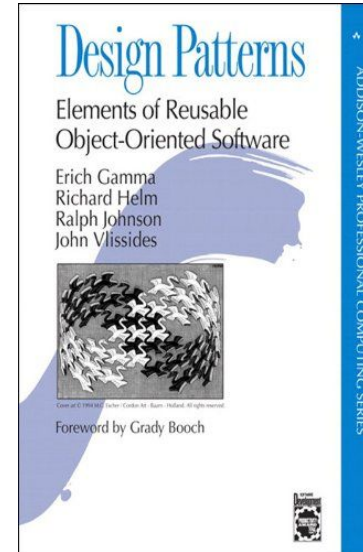
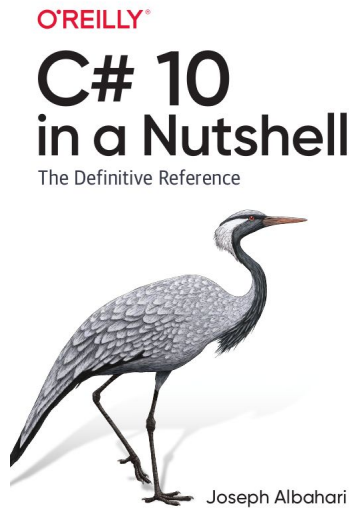
- **Monitor:** Pedro Ciambra
  - [pedro.ciambra@ic.unicamp.br](mailto:pedro.ciambra@ic.unicamp.br)
- **Aulas**
  - Carga horária: 16H – 4 aulas
  - Localização: IC3
  - 24/09, 01/10 e 08/10
    - 8:30-10h30: Aula na sala 351
    - 10h30-12h30: Laboratório nas salas 303 e 304
  - 15/10
    - 8h30-12h30: Aula e laboratório na sala 300



- I/O, File e database
  - Streams, XML and JSON, LINQ
- Parallel programming
  - Threading, Tasks and Synchronization
- Design patterns
  - GoF, tipo de design patterns
- .NET Core, .NET Standard and .NET Framework
  - Signing and deploying assemblies



- 4 listas de trabalhos práticos
  - em dupla (ou individual)
  - desenvolvimento no Github
  - entrega no Moodle
  - prazo de 7 dias
- Nota final
  - = média aritmética das notas das listas



- Joseph Albahari.  
C# 10 in a Nutshell: The Definitive Reference.  
2022.
- E. Gamma and R. Helm and R. Johnson and J. Vlissides.  
Design Patterns - Elements of Reusable Object-Oriented Software.  
1995.



- I/O e Arquivos
  - Arquivos texto e binário
  - Stream
  - Compressão
  - Gestão de arquivos e diretórios
- Arquivos XML e JSON
  - Document Object Model
- LINQ
  - Linguagem de consulta
  - Consulta em database/arquivos
  - Operadores LINQ

The background features a network of gray lines connecting various colored circles (orange, yellow, blue, green) and a pattern of small, faint circuit-like symbols on a dark blue horizontal band.

# I/O e Arquivos



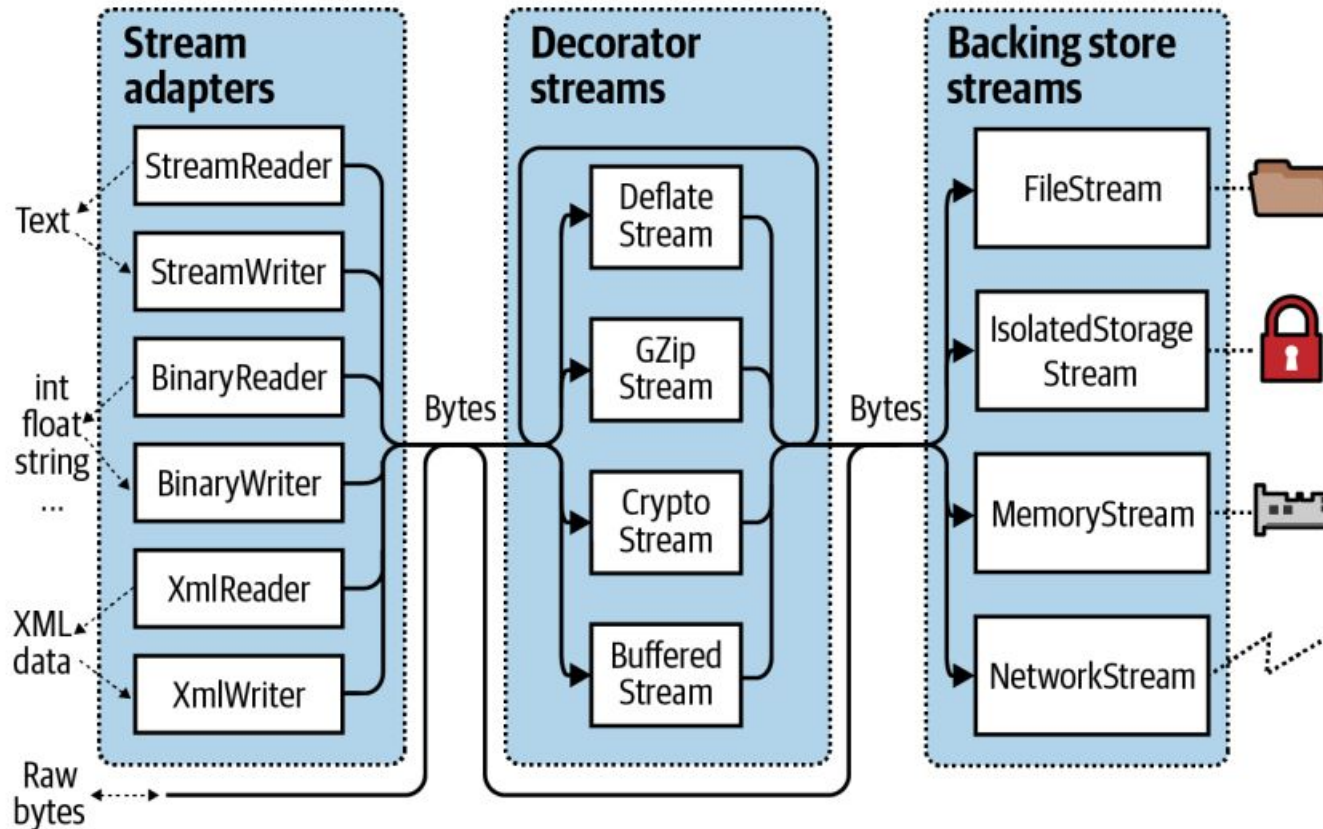


- Arquivos podem ter o mais variado conteúdo, mas do ponto de vista dos programas existem apenas dois tipos de arquivos:
  - **Arquivo texto:** Armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de textos simples.  
Exemplos: código fonte C#, documento texto simples, páginas HTML (HyperText Markup Language), arquivos CSV (Comma-Separated Values).
  - **Arquivo binário:** Sequência de bits sujeita às convenções do programa que o gerou, não legíveis diretamente por um humano.  
Exemplos: arquivos executáveis, arquivos compactados, documentos do Word.



- A arquitetura de Stream do .NET
  - Fornece uma interface de programação consistente para leitura e escrita em vários tipos de I/O
- Classes para manipular arquivos e diretórios no disco
- Fluxos especializados para compactação, pipes nomeados e arquivos mapeados na memória
- Vamos nos concentrar no namespace System.IO
  - Funcionalidade de E/S de baixo nível

# Arquitetura do Stream



# Membros da classe Stream



Category	Members
Reading	<pre>public abstract bool CanRead { get; } public abstract int Read (byte[] buffer, int offset, int count) public virtual int ReadByte();</pre>
Writing	<pre>public abstract bool CanWrite { get; } public abstract void Write (byte[] buffer, int offset, int count); public virtual void WriteByte (byte value);</pre>
Seeking	<pre>public abstract bool CanSeek { get; } public abstract long Position { get; set; } public abstract void SetLength (long value); public abstract long Length { get; } public abstract long Seek (long offset, SeekOrigin origin);</pre>
Closing/ flushing	<pre>public virtual void Close();  public void Dispose(); public abstract void Flush();</pre>
Timeouts	<pre>public virtual bool CanTimeout { get; } public virtual int ReadTimeout { get; set; } public virtual int WriteTimeout { get; set; }</pre>
Other	<pre>public static readonly Stream Null; // "Null" stream public static Stream Synchronized (Stream stream);</pre>

# Primeiro Exemplo



```
using System;
using System.IO;

// Create a file called test.txt in the current directory:
using (Stream s = new FileStream ("test.txt", FileMode.Create))
{
    Console.WriteLine (s.CanRead);           // True
    Console.WriteLine (s.CanWrite);          // True
    Console.WriteLine (s.CanSeek);           // True

    s.WriteByte (101);
    s.WriteByte (102);
    byte[] block = { 1, 2, 3, 4, 5 };
    s.Write (block, 0, block.Length);         // Write block of 5 bytes

    Console.WriteLine (s.Length);             // 7
    Console.WriteLine (s.Position);           // 7
    s.Position = 0;                          // Move back to the start

    Console.WriteLine (s.ReadByte());          // 101
    Console.WriteLine (s.ReadByte());          // 102

    // Read from the stream back into the block array:
    Console.WriteLine (s.Read (block, 0, block.Length)); // 5

    // Assuming the last Read returned 5, we'll be at
    // the end of the file, so Read will now return 0:
    Console.WriteLine (s.Read (block, 0, block.Length)); // 0
}
```

# Leitura Direta da Stream



```
public abstract bool CanRead { get; }  
public abstract int Read (byte[] buffer, int offset, int count)  
public virtual int ReadByte();
```

```
// Assuming s is a stream:  
byte[] data = new byte [1000];  
s.Read (data, 0, data.Length);
```

Pode não ler tudo

Forma  
Correta

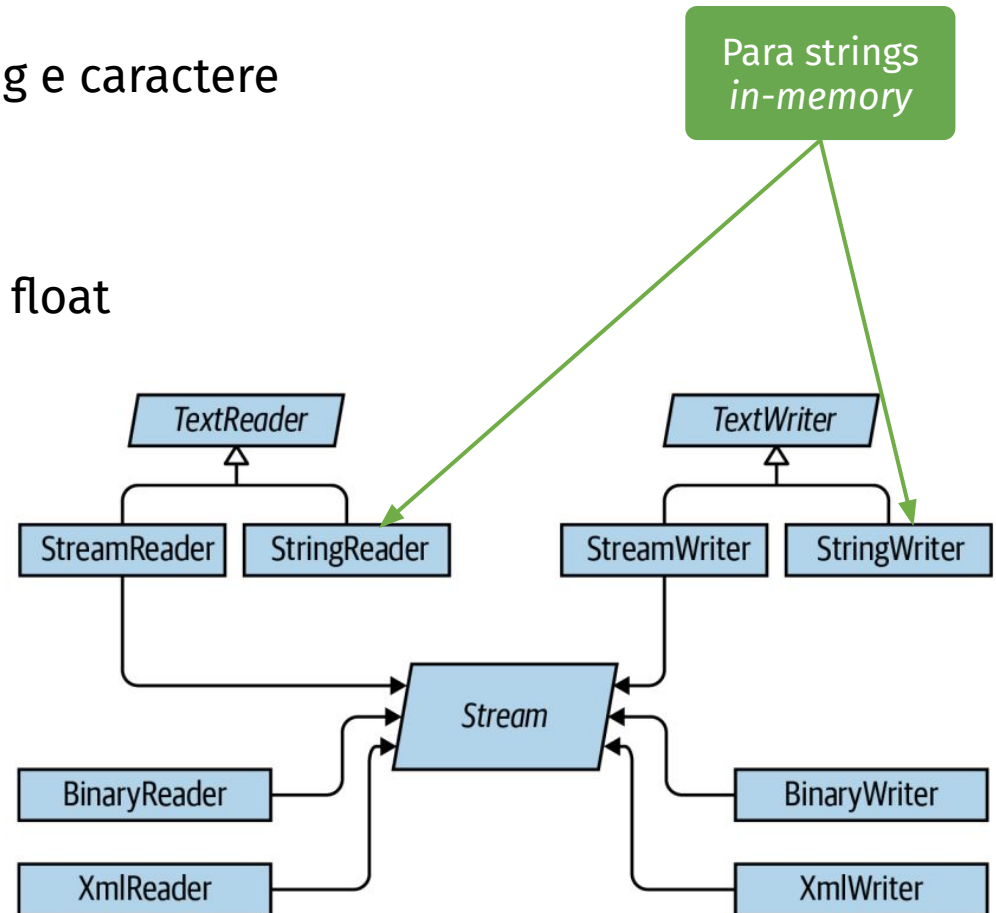
```
byte[] data = new byte [1000];  
  
// bytesRead will always end up at 1000, unless the stream is  
// itself smaller in length:  
  
int bytesRead = 0;  
int chunkSize = 1;  
while (bytesRead < data.Length && chunkSize > 0)  
    bytesRead +=  
        chunkSize = s.Read (data, bytesRead, data.Length - bytesRead);
```

Muito  
baixo nível !!

# Stream Adapters



- Text adapters
  - Para dados de tipo string e caractere
- Binary adapters
  - Para tipos primitivos
  - Como int, bool, string, e float
- XML adapters
  - Para arquivos XML





# Membros da Classe TextReader



Category	Members
Reading one char	<code>public virtual int Peek(); // Cast the result to a char</code> <code>public virtual int Read(); // Cast the result to a char</code>
Reading many chars	<code>public virtual int Read (char[] buffer, int index, int count);</code> <code>public virtual int ReadBlock (char[] buffer, int index, int count);</code> <code>public virtual string ReadLine();</code> <code>public virtual string ReadToEnd();</code>
Closing	<code>public virtual void Close();</code> <code>public void Dispose(); // Same as Close</code>
Other	<code>public static readonly TextReader Null;</code> <code>public static TextReader Synchronized (TextReader reader);</code>



# Membros da Classe TextWriter



Category	Members
Writing one char	<code>public virtual void Write (char value);</code>
Writing many chars	<code>public virtual void Write (string value);</code> <code>public virtual void Write (char[] buffer, int index, int count);</code> <code>public virtual void Write (string format, params object[] arg);</code> <code>public virtual void WriteLine (string value);</code>
Closing and flushing	<code>public virtual void Close();</code>  <code>public void Dispose(); // Same as Close</code> <code>public virtual void Flush();</code>
Formatting and encoding	<code>public virtual IFormatProvider FormatProvider { get; }</code>  <code>public virtual string NewLine { get; set; }</code> <code>public abstract Encoding Encoding { get; }</code>
Other	<code>public static readonly TextWriter Null;</code> <code>public static TextWriter Synchronized (TextWriter writer);</code>

# Exemplo usando Text Adapters



```
using (FileStream fs = File.Create ("test.txt"))
using (TextWriter writer = new StreamWriter (fs))
{
    writer.WriteLine ("Line1");
    writer.WriteLine ("Line2");
}

using (FileStream fs = File.OpenRead ("test.txt"))
using (TextReader reader = new StreamReader (fs))
{
    Console.WriteLine (reader.ReadLine());           // Line1
    Console.WriteLine (reader.ReadLine());           // Line2
}
```

# Exemplo usando Text Adapters (2)



A classe File fornece métodos estáticos como atalhos

```
using (TextWriter writer = File.CreateText ("test.txt"))
{
    writer.WriteLine ("Line1");
    writer.WriteLine ("Line2");
}

using (TextWriter writer = File.AppendText ("test.txt"))
    writer.WriteLine ("Line3");

using (TextReader reader = File.OpenText ("test.txt"))
    while (reader.Peek() > -1)
        Console.WriteLine (reader.ReadLine());           // Line1
                                                         // Line2
                                                         // Line3
```

# Exemplo usando Text Adapters (3)



Pode usar para ler/escrever outros tipos também

```
using (TextWriter w = File.CreateText ("data.txt"))
{
    w.WriteLine (123);           // Writes "123"
    w.WriteLine (true);          // Writes the word "true"
}


using (TextReader r = File.OpenText ("data.txt"))
{
    int myInt = int.Parse (r.ReadLine());    // myInt == 123
    bool yes = bool.Parse (r.ReadLine());    // yes == true
}
```

# Exemplo usando Binary Adapters



```
public class Person
{
    public string Name;
    public int    Age;
    public double Height;
}
```

Formatos binários  
usam em geral  
menos memória  
que texto



```
public void SaveData (Stream s)
{
    var w = new BinaryWriter (s);
    w.Write (Name);
    w.Write (Age);
    w.Write (Height);
    w.Flush();           // Ensure the BinaryWriter buffer is cleared.
                        // We won't dispose/close it, so more data
                        // can be written to the stream.
}
```

```
public void LoadData (Stream s)
{
    var r = new BinaryReader (s);
    Name   = r.ReadString();
    Age    = r.ReadInt32();
    Height = r.ReadDouble();
}
```

# Fechar o Adapter



1. Fecha somente o adapter
2. Fecha o adapter e depois fecha a stream
3. (Writers) Flush o adapter e depois fecha a stream
4. (Readers) Fecha a stream

Nunca fecha a stream **antes** de fechar ou flushear o writer!!

# Streams de Compressão



- Namespace
  - System.IO.Compression
- Streams de compressão de uso geral
  - DeflateStream
  - GZipStream
  - BrotliStream -> taxa de compressão mais alta, mas mais lenta
- 2 operações
  - *write* no stream para comprimir
  - *read* no stream para descomprimir



- Streams de compressão se usam como decorator
  - Tem que passar uma stream normal no construtor

```
using (Stream s = File.Create ("compressed.bin"))
using (Stream ds = new DeflateStream (s, CompressionMode.Compress))
    for (byte i = 0; i < 100; i++)
        ds.WriteByte (i);

using (Stream s = File.OpenRead ("compressed.bin"))
using (Stream ds = new DeflateStream (s, CompressionMode.Decompress))
    for (byte i = 0; i < 100; i++)
        Console.WriteLine (ds.ReadByte());    // Writes 0 to 99
```



# Compressão *In-memory*



```
byte[] data = new byte[1000];           // We can expect a good compression
                                         // ratio from an empty array!

var ms = new MemoryStream();
using (Stream ds = new DeflateStream (ms, CompressionMode.Compress))
    ds.Write (data, 0, data.Length);

byte[] compressed = ms.ToArray();
Console.WriteLine (compressed.Length);    // 11

// Decompress back to the data array:
ms = new MemoryStream (compressed);
using (Stream ds = new DeflateStream (ms, CompressionMode.Decompress))
    for (int i = 0; i < 1000; i += ds.Read (data, i, 1000 - i));
```

MemoryStream oferece as  
funcionalidades das  
Streams sem usar  
arquivos



- Vantagem do formato ZIP
  - Container que tem suporte a múltiplos arquivos
- 2 classes disponíveis
  - ZipArchive para usar com Stream
  - ZipFile para trabalhar diretamente com arquivos

```
ZipFile.CreateFromDirectory (@":\MyFolder", @":\archive.zip");
```

```
ZipFile.ExtractToDirectory (@":\archive.zip", @":\MyFolder");
```

```
using (ZipArchive zip = ZipFile.Open (@":\zz.zip", ZipArchiveMode.Read))  
  
    foreach (ZipArchiveEntry entry in zip.Entries)  
        Console.WriteLine (entry.FullName + " " + entry.Length);
```



- File, Directory e Path
  - Classes estáticas
- FileInfo e DirectoryInfo
  - Classes construídas com o nome do arquivo ou o directório



```
bool Exists (string path);           // Returns true if the file is present

void Delete  (string path);
void Copy    (string sourceFileName, string destFileName);
void Move    (string sourceFileName, string destFileName);
void Replace (string sourceFileName, string destinationFileName,
               string destinationBackupFileName);

FileAttributes GetAttributes (string path);
void SetAttributes          (string path, FileAttributes fileAttributes);

void Decrypt (string path);
void Encrypt (string path);

DateTime GetCreationTime  (string path);           // UTC versions are
DateTime GetLastAccessTime (string path);           // also provided.
DateTime GetLastWriteTime (string path);

void SetCreationTime  (string path, DateTime creationTime);
void SetLastAccessTime (string path, DateTime lastAccessTime);
void SetLastWriteTime (string path, DateTime lastWriteTime);

FileSecurity GetAccessControl (string path);
FileSecurity GetAccessControl (string path,
                               AccessControlSections includeSections);
void SetAccessControl (string path, FileSecurity fileSecurity);
```



```
string GetCurrentDirectory ();  
void    SetCurrentDirectory (string path);  
  
DirectoryInfo CreateDirectory  (string path);  
DirectoryInfo GetParent       (string path);  
string       GetDirectoryRoot (string path);  
  
string[] GetLogicalDrives(); // Gets mount points on Unix  
  
// The following methods all return full paths:  
  
string[] GetFiles           (string path);  
string[] GetDirectories     (string path);  
string[] GetFileSystemEntries (string path);  
  
IEnumerable<string> EnumerateFiles           (string path);  
IEnumerable<string> EnumerateDirectories     (string path);  
IEnumerable<string> EnumerateFileSystemEntries (string path);
```



Expression	Result (Windows, then Unix)
Directory.GetCurrentDirectory()	k:\demo\ or /demo
Path.IsPathRooted (file)	False
Path.IsPathRooted (path)	True
Path.GetPathRoot (path)	c:\ or /
Path.GetDirectoryName (path)	c:\mydir or /mydir
Path.GetFileName (path)	myfile.txt
Path.GetFullPath (file)	k:\demo\myfile.txt or /demo/myfile.txt
Path.Combine (dir, file)	c:\mydir\myfile.txt or /mydir/myfile.txt
<b>File extensions:</b>	
Path.HasExtension (file)	True
Path.GetExtension (file)	.txt
Path.GetFileNameWithoutExtension (file)	myfile
Path.ChangeExtension (file, ".log")	myfile.log
<b>Separators and characters:</b>	
Path.DirectorySeparatorChar	\ or /
Path.AltDirectorySeparatorChar	/
Path.PathSeparator	; or :
Path.VolumeSeparatorChar	: or /
Path.GetInvalidPathChars()	chars 0 to 31 and "<> eor 0
Path.GetInvalidFileNameChars()	chars 0 to 31 and "<> :*?\  or 0 and /
<b>Temporary files:</b>	
Path.GetTempPath()	<local user folder>\Temp or /tmp/
Path.GetRandomFileName()	d2dwuzjf.dnp
Path.GetTempFileName()	<local user folder>\Temp\tmp14B.tmp or /tmp/tmpubSUYO.tmp

The background features a network of gray lines connecting various colored circles (orange, yellow, blue, green) and rectangular blocks. A dark blue horizontal band with a subtle circuit pattern is positioned across the middle, containing the title text. The overall aesthetic is modern and technical.

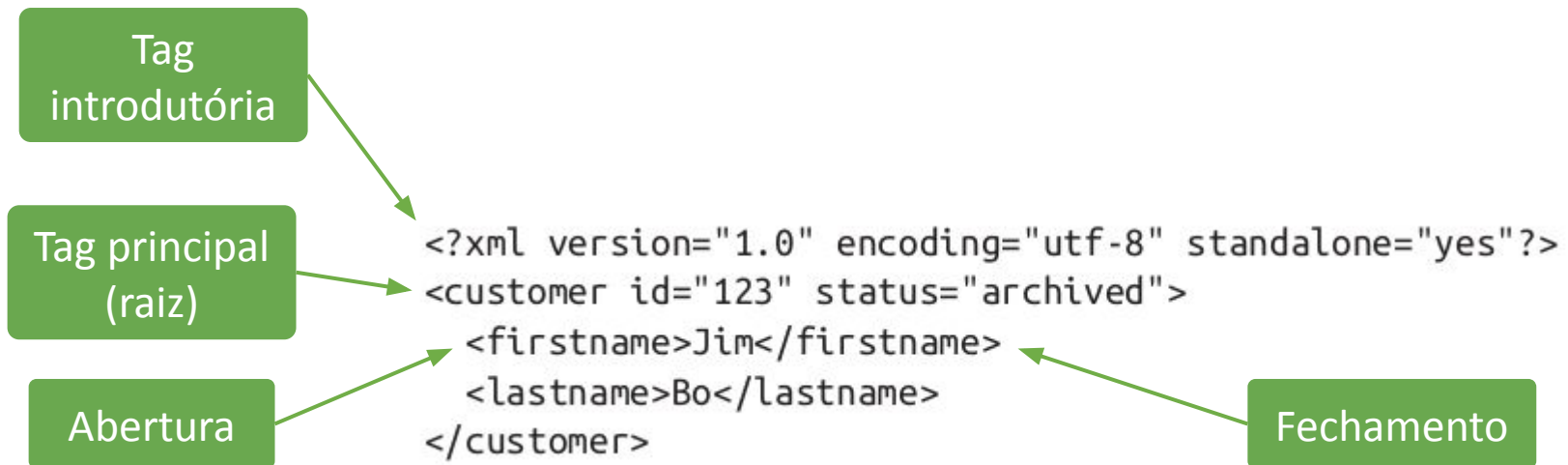
# Arquivos XML





- **eXtensible Markup Language**

- uma linguagem de marcação usada para criação de documentos com dados organizados de maneira hierárquica
- exemplo: Nota Fiscal Eletrônica (NFe), docx (comprimido)








```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<customer id="123" status="archived">
  <firstname>Jim</firstname>
  <lastname>Bo</lastname>
</customer>
```

```
XmlReaderSettings settings = new XmlReaderSettings();
settings.IgnoreWhitespace = true;

using XmlReader reader = XmlReader.Create ("customer.xml", settings);
while (reader.Read())
{
    Console.Write (new string (' ', reader.Depth * 2)); // Write indentation
    Console.Write (reader.NodeType.ToString());

    if (reader.NodeType == XmlNodeType.Element ||
        reader.NodeType == XmlNodeType.EndElement)
    {
        Console.Write (" Name=" + reader.Name);
    }
    else if (reader.NodeType == XmlNodeType.Text)
    {
        Console.Write (" Value=" + reader.Value);
    }
    Console.WriteLine ();
}
```



XmlDeclaration  
Element Name=customer  
  Element Name=firstname  
    Text Value=Jim  
  EndElement Name=firstname  
  Element Name=lastname  
    Text Value=Bo  
  EndElement Name=lastname  
EndElement Name=customer

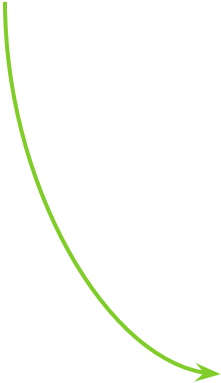
# Métodos *Read*



Members	Works on NodeType	Sample XML fragment	Input parameters	Data returned
ReadContentAsXXX	Text	<a>x</a>		x
ReadElementContentAsXXX	Element	<a>x</a>		x
ReadInnerXml	Element	<a>x</a>		x
ReadOuterXml	Element	<a>x</a>		<a>x</a>
ReadStartElement	Element	<a>x</a>		
ReadEndElement	Element	<a>x</a>		
ReadSubtree	Element	<a>x</a>		<a>x</a>
ReadToDescendant	Element	<a>x<b></b></a>	"b"	
ReadToFollowing	Element	<a>x<b></b></a>	"b"	
ReadToNextSibling	Element	<a>x</a><b></b>	"b"	
ReadAttribute Value	Attribute	See <i>"Reading Attributes"</i> on page 535		



```
XmlWriterSettings settings = new XmlWriterSettings();  
settings.Indent = true;  
  
using XmlWriter writer = XmlWriter.Create ("foo.xml", settings);  
  
writer.WriteStartElement ("customer");  
writer.WriteElementString ("firstname", "Jim");  
writer.WriteElementString ("lastname", "Bo");  
writer.WriteEndElement();
```

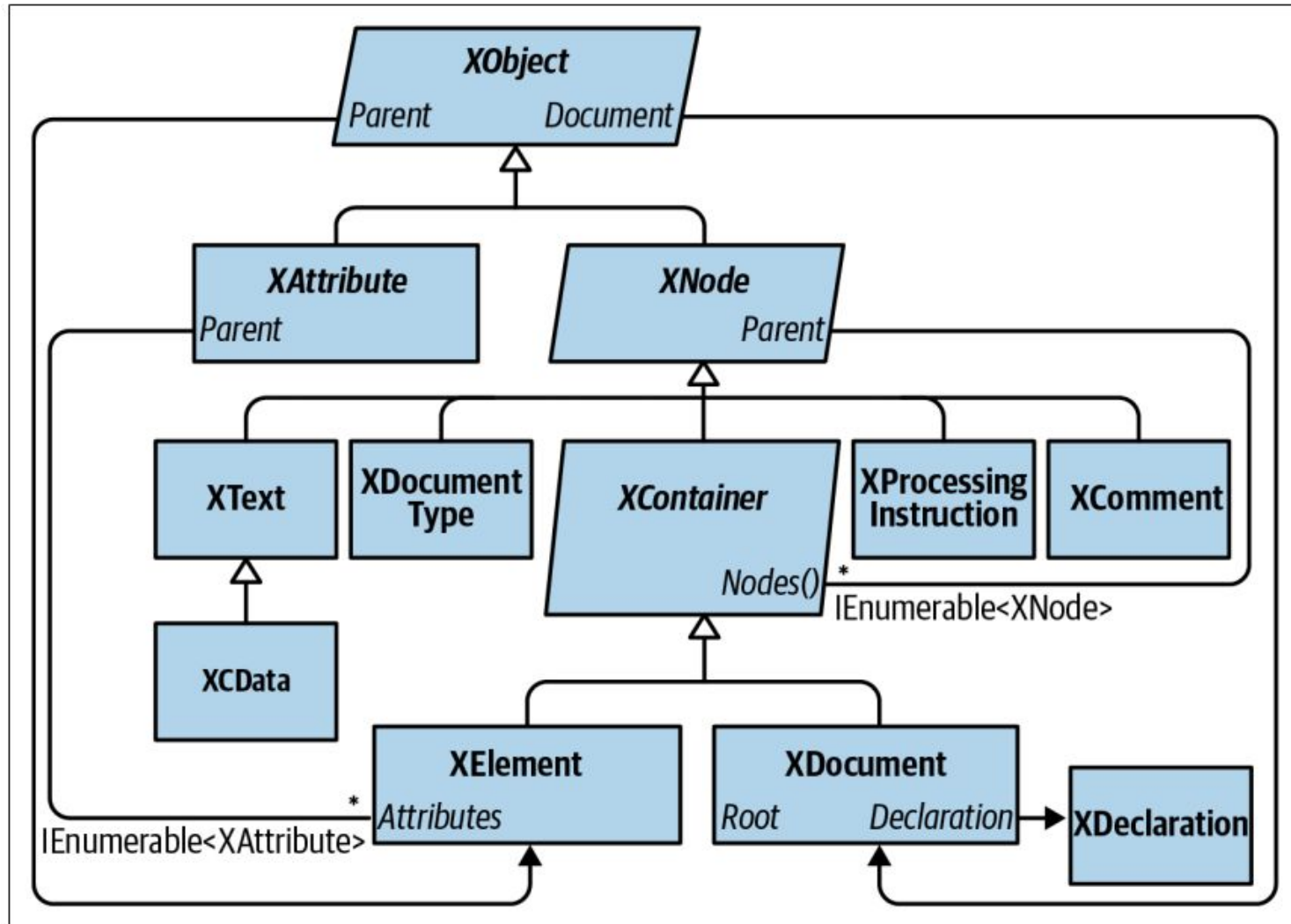


```
<?xml version="1.0" encoding="utf-8"?>  
<customer>  
  <firstname>Jim</firstname>  
  <lastname>Bo</lastname>  
</customer>
```



- **Modelo de Documento por Objetos**
  - Convenção multiplataforma e independente de linguagem de programação
  - Para representação e interação com objetos em documentos HTML, XHTML e XML
  - Onde os elementos/nós de cada documento são organizados em uma estrutura de árvore
  - Definindo métodos para que possam ser alterados estrutura, estilo e conteúdo do documento

# XML DOM (X-DOM)

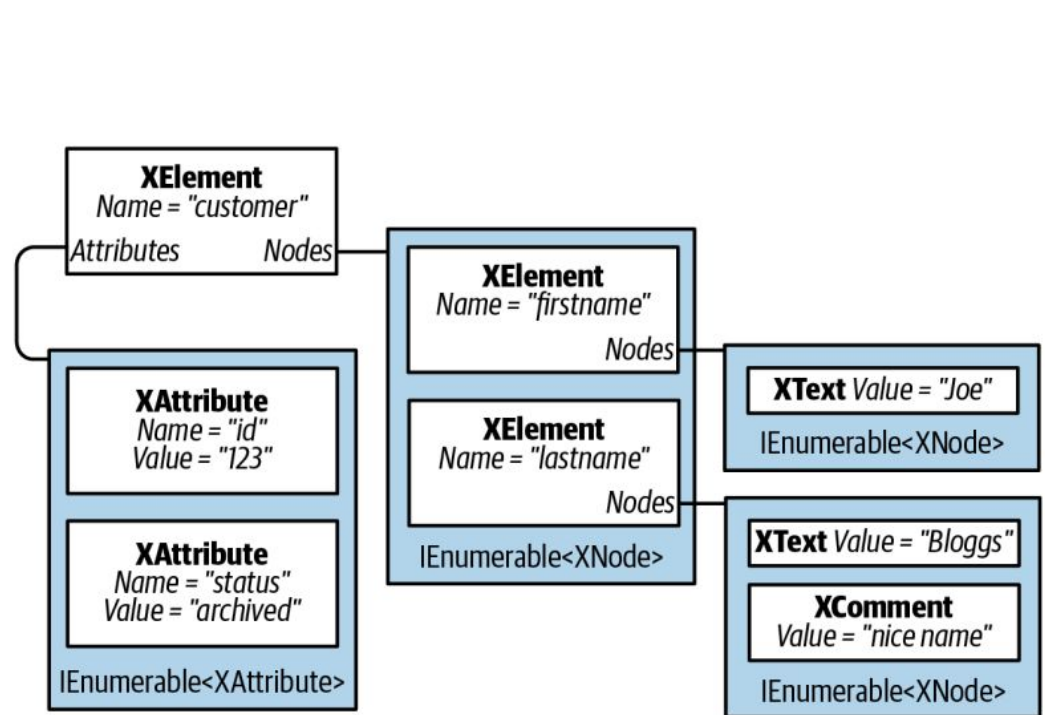


# Exemplo de árvore X-DOM



```
string xml = @"<customer id='123' status='archived'>
    <firstname>Joe</firstname>
    <lastname>Bloggs<!-- nice name --></lastname>
</customer>";
```

```
XElement customer = XElement.Parse (xml);
```



# Carregar e parsear um X-DOM



```
XDocument fromWeb = XDocument.Load ("http://albahari.com/sample.xml");
```

```
XElement fromFile = XElement.Load (@":\\media\\somefile.xml");
```

```
XElement config = XElement.Parse (  
@"<configuration>  
  <client enabled='true'>  
    <timeout>30</timeout>  
  </client>  
</configuration>");
```

**XDocument**  
é o **XElement**  
da Raiz  
(uso facultativo)



# Instanciar um X-DOM



```
XElement lastName = new XElement ("lastname", "Bloggs");  
lastName.Add (new XComment ("nice name"));
```

```
XElement customer = new XElement ("customer");  
customer.Add (new XAttribute ("id", 123));  
customer.Add (new XElement ("firstname", "Joe"));  
customer.Add (lastName);
```

```
Console.WriteLine (customer.ToString());
```

```
<customer id="123">  
  <firstname>Joe</firstname>  
  <lastname>Bloggs<!-- nice name --></lastname>  
</customer>
```

```
XElement customer =  
    new XElement ("customer", new XAttribute ("id", 123),  
        new XElement ("firstname", "joe"),  
        new XElement ("lastname", "bloggs",  
            new XComment ("nice name")  
        )  
    );
```

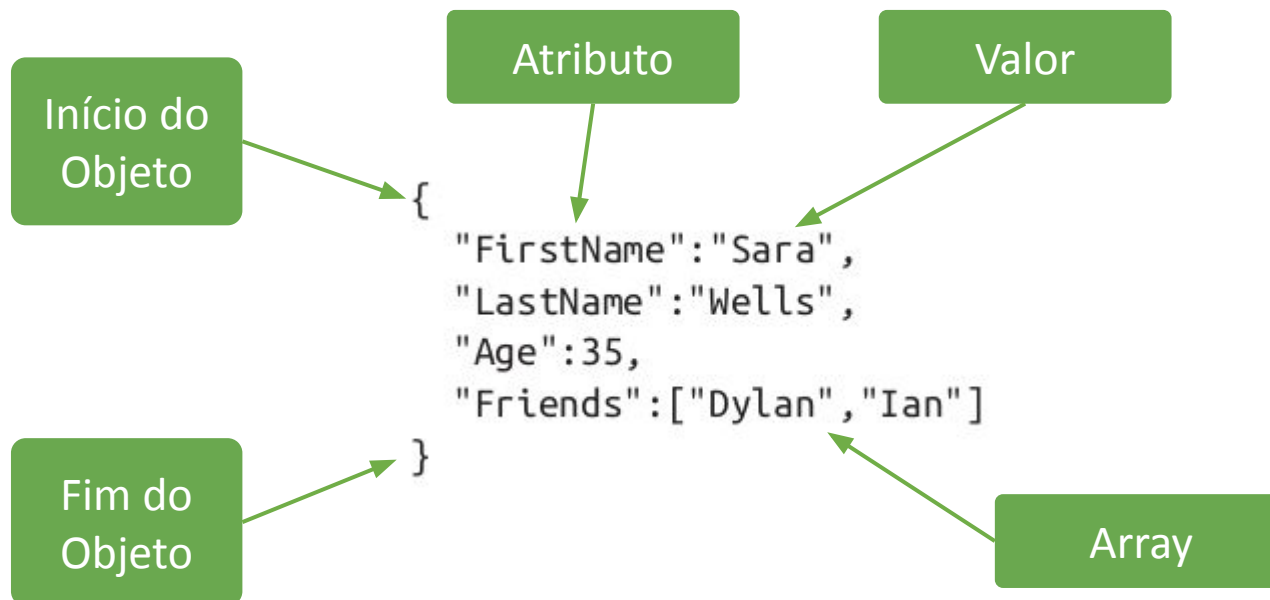


The background features a network of gray lines connecting various colored circles (orange, yellow, blue, green) and rectangular bars (orange, blue, green). A dark blue horizontal band with a subtle circuit pattern is positioned behind the title.

# Arquivos JSON



- **JSON** (*JavaScript Object Notation*)
  - Muito usado hoje
- Suporte varias tipos de dados
  - **Simples:** numero, string, boolean,
  - **Complexos:** objeto (coleção atributo valor), array



# JsonReader



```
byte[] data = File.ReadAllBytes ("people.json");
Utf8JsonReader reader = new Utf8JsonReader (data);
while (reader.Read())
{
    switch (reader.TokenType)
    {
        case JsonTokenType.StartObject:
            Console.WriteLine ($"Start of object");
            break;
        case JsonTokenType.EndObject:
            Console.WriteLine ($"End of object");
            break;
        case JsonTokenType.StartArray:
            Console.WriteLine();
            Console.WriteLine ($"Start of array");
            break;
        case JsonTokenType.EndArray:
            Console.WriteLine ($"End of array");
            break;
        case JsonTokenType.PropertyName:
            Console.Write ($"Property: {reader.GetString()}");
            break;
        case JsonTokenType.String:
            Console.WriteLine ($" Value: {reader.GetString()}");
            break;
        case JsonTokenType.Number:
            Console.WriteLine ($" Value: {reader.GetInt32()}");
            break;
        default:
            Console.WriteLine ($"No support for {reader.TokenType}");
            break;
    }
}
```

```
{
  "FirstName": "Sara",
  "LastName": "Wells",
  "Age": 35,
  "Friends": ["Dylan", "Ian"]
}
```




Start of object  
Property: FirstName Value: Sara  
Property: LastName Value: Wells  
Property: Age Value: 35  
Property: Friends  
Start of array  
Value: Dylan  
Value: Ian  
End of array  
End of object



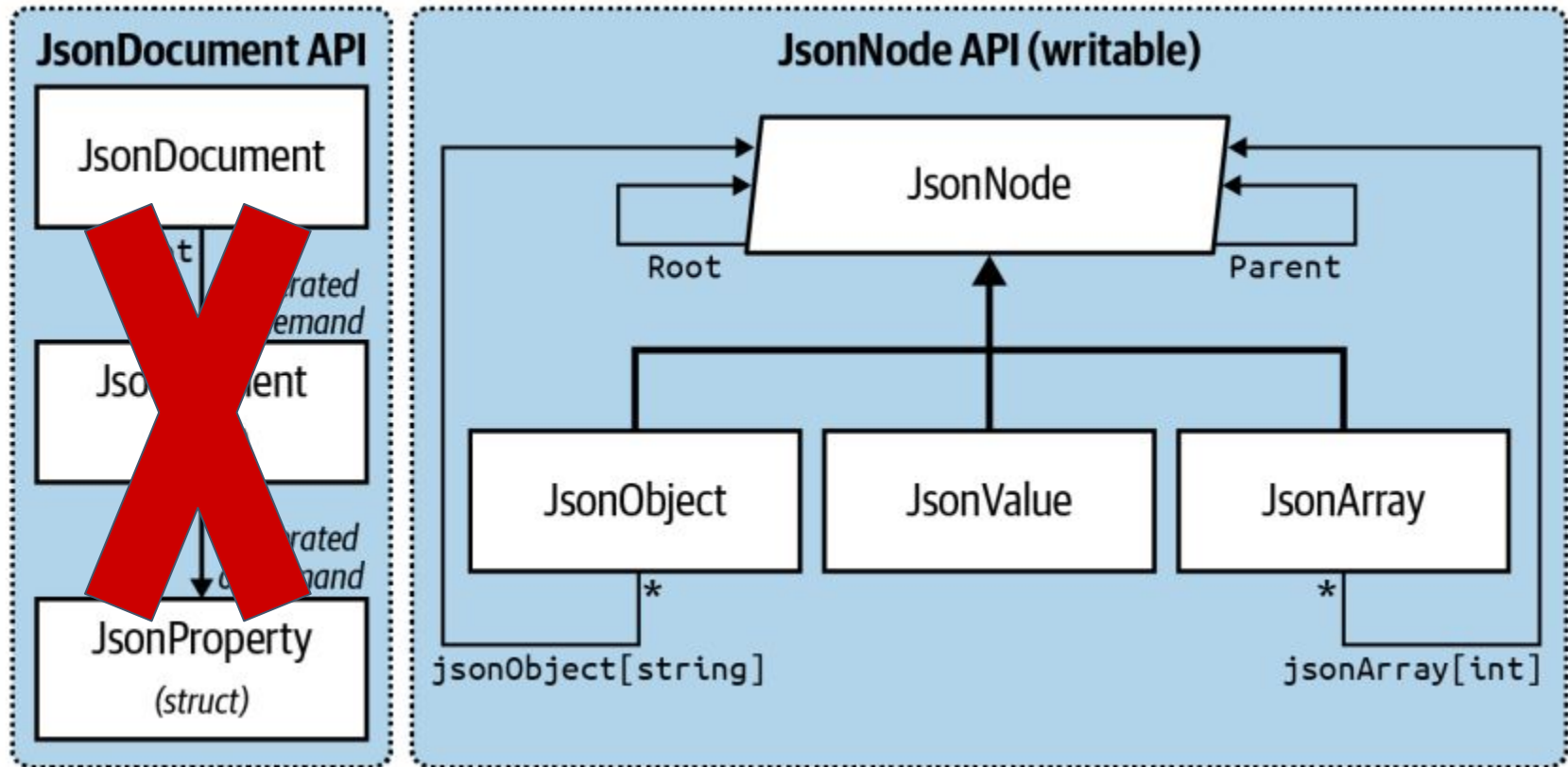
```
var options = new JsonWriterOptions { Indented = true };

using (var stream = File.Create ("MyFile.json"))
using (var writer = new Utf8JsonWriter (stream, options))
{
    writer.WriteStartObject();
    // Property name and value specified in one call
    writer.WriteString ("FirstName", "Dylan");
    writer.WriteString ("LastName", "Lockwood");
    // Property name and value specified in separate calls
    writer.WritePropertyName ("Age");
    writer.WriteNumberValue (46);
    writer.WriteCommentValue ("This is a (non-standard) comment");
    writer.WriteEndObject();
}
```



```
{
    "FirstName": "Dylan",
    "LastName": "Lockwood",
    "Age": 46
    /*This is a (non-standard) comment*/
}
```

# JSON DOM APIs



# Ler valor simples com JsonNode



```
var node = JsonNode.Parse ("123"); // Parses to a JsonValue  
int number = node.AsValue().GetValue<int>();  
// Shortcut for ((JsonValue)node).GetValue<int>();
```



```
var node = JsonNode.Parse ("123");  
int number = node.GetValue<int>();  
// Shortcut for node.AsValue().GetValue<int>();
```



```
var node = JsonNode.Parse ("123");  
int number = (int) node;
```



```
if (node.AsValue().TryGetValue<int> (out var number))  
    Console.WriteLine (number);
```

Quando não tem  
certeza que o  
parsing vai  
funcionar

# Ler array com JsonNode



```
var node = JsonNode.Parse(@"[1, 2, 3, 4, 5]");  
Console.WriteLine (node.AsArray().Count);           // 5  
  
foreach (JsonNode child in node.AsArray())  
{ ... }
```

JSON array  
implementa  
IList<JsonNode>

```
Console.WriteLine ((int)node[0]); // 1
```

Shortcut



# Ler object com JsonNode



```
var node = JsonNode.Parse (@"{ \"Name\":\"Alice\", \"Age\": 32}");  
string name = (string) node ["Name"];    // Alice  
int age = (int) node ["Age"];             // 32
```

**JsonObject**  
implementa  
**IDictionary<string,JsonNode>**

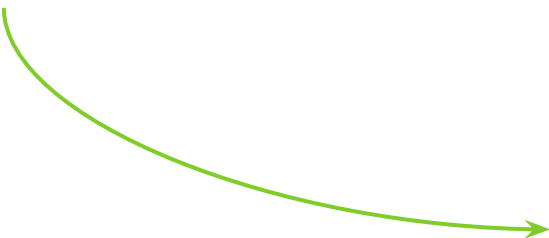
```
// Enumerate over the dictionary's key/value pairs:  
foreach (KeyValuePair<string,JsonNode> keyValuePair in node.AsObject())  
{  
    string propertyName = keyValuePair.Key;    // "Name" (then "Age")  
    JsonNode value = keyValuePair.Value;  
}
```



# Construir com JsonNode



```
var node = new JsonArray
{
    new JsonObject {
        ["Name"] = "Tracy",
        ["Age"] = 30,
        ["Friends"] = new JsonArray ("Lisa", "Joe")
    },
    new JsonObject {
        ["Name"] = "Jordyn",
        ["Age"] = 25,
        ["Friends"] = new JsonArray ("Tracy", "Li")
    }
};
```



```
[
  {
    "Name": "Tracy",
    "Age": 30,
    "Friends": ["Lisa", "Joe"]
  },
  {
    "Name": "Jordyn",
    "Age": 25,
    "Friends": ["Tracy", "Li"]
  }
]
```

The background features a network of gray lines connecting various colored circles (orange, yellow, blue, green) and a dark blue horizontal band with a subtle circuit pattern.

# Introdução ao LINQ



- Language Integrated Query (LINQ)
  - Linguagens e *runtime feature* para escrever consultas estruturadas *type-safe* em coleções de objetos locais e fontes de dados remotas
  - 2 sintaxe possíveis (fluent e query)
- Permite consultar qualquer coleção
  - matriz, lista ou arquivo XML/JSON (DOM)
  - tabelas em um banco de dados SQL
- Benefícios
  - verificação de tipo em tempo de compilação
  - composição de consulta dinâmica



- É comum trabalhar com um conjunto de objetos
  - Uma coleção é um objeto que agrupa múltiplos elementos (variáveis primitivas ou objetos) dentro de uma única unidade,
- Coleções em C#
  - Interfaces que definem protocolos padrões para coleções
    - IEnumerable, IEnumerator, ICollection, IList
  - Classes de coleção prontas para uso
    - Lista, queue, pilha, dicionário, etc
  - Classes base para implementar coleções específicas
    - CollectionBase, Collection<T>

# Fluent Syntax



```
string[] names = { "Tom", "Dick", "Harry" };  
IEnumerable<string> filteredNames = System.Linq.Enumerable.Where  
    (names, n => n.Length >= 4);  
foreach (string n in filteredNames)  
    Console.WriteLine (n);
```

Use *lambda expressions*

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
  
string[] names = { "Tom", "Dick", "Harry" };  
  
IEnumerable<string> filteredNames = names.Where (n => n.Length >= 4);  
foreach (string name in filteredNames) Console.WriteLine (name);
```

```
var filteredNames = names.Where (n => n.Length >= 4);
```

# Operadores encadeados

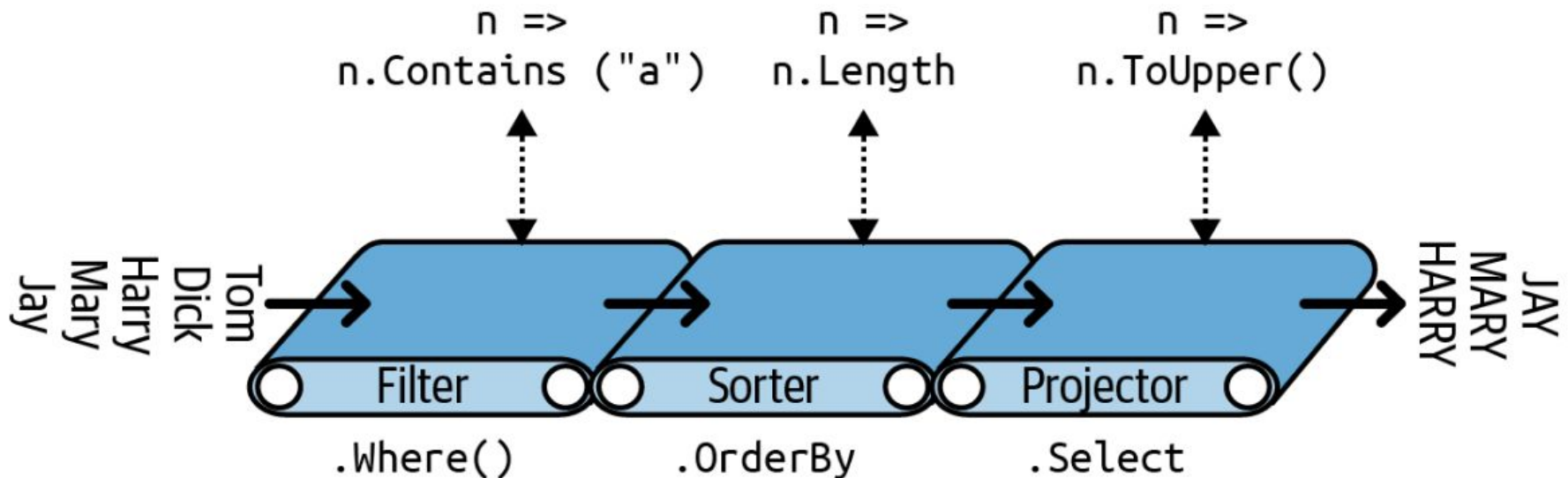


```
using System;
using System.Collections.Generic;
using System.Linq;

string[] names = { "Tom", "Dick", "Harry", "Mary", "Jay" };

IEnumerable<string> query = names
    .Where (n => n.Contains ("a"))
    .OrderBy (n => n.Length)
    .Select (n => n.ToUpper());

foreach (string name in query) Console.WriteLine (name);
```



# Query Syntax



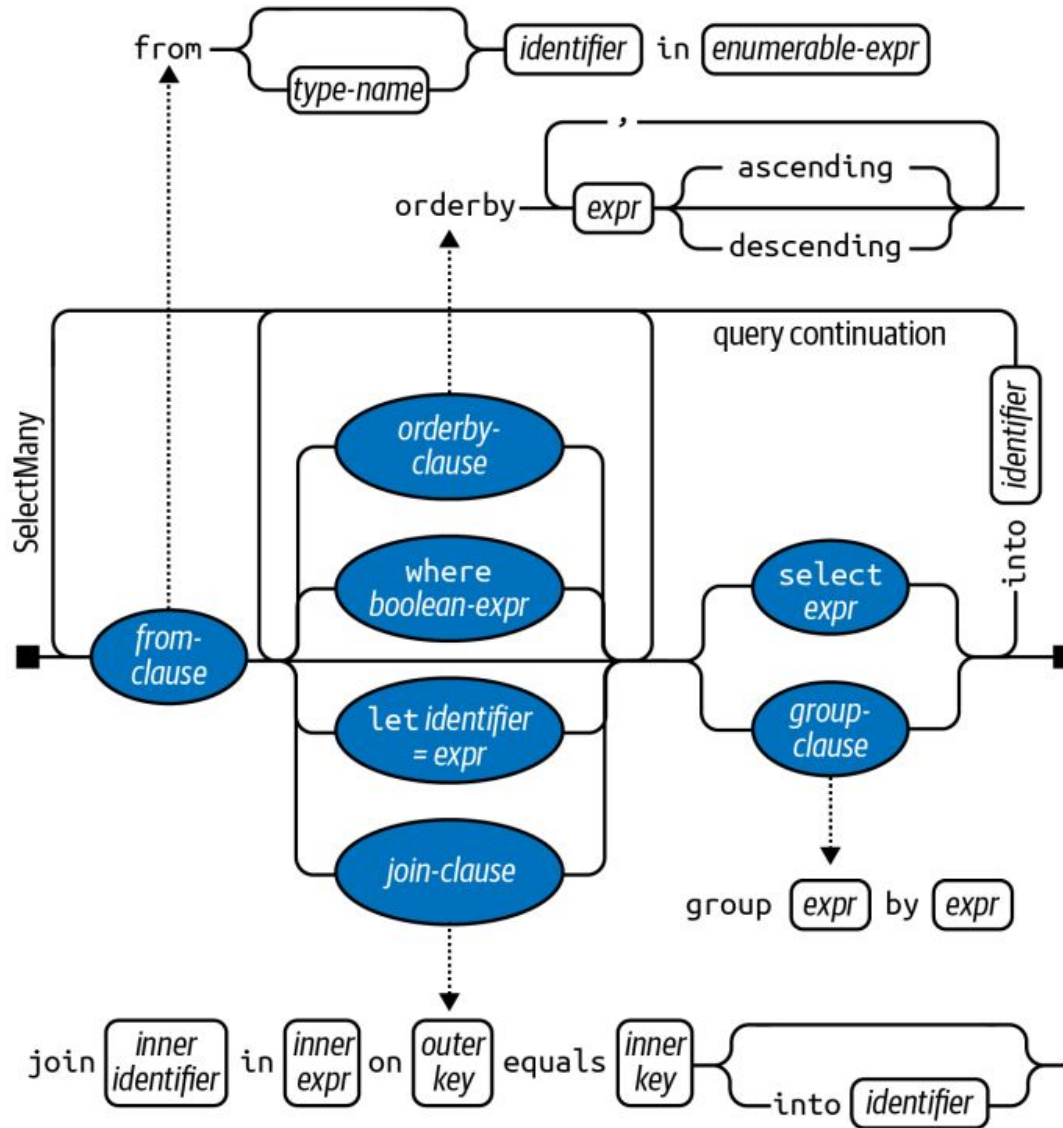
```
using System;
using System.Collections.Generic;
using System.Linq;

string[] names = { "Tom", "Dick", "Harry", "Mary", "Jay" };

IEnumerable<string> query =
    from    n in names
    where   n.Contains ("a")      // Filter elements
    orderby n.Length              // Sort elements
    select  n.ToUpper();         // Translate each element (project)

foreach (string name in query) Console.WriteLine (name);
```

# Query Syntax







- Operador query não são executado na construção
  - *deferred or lazy execution*

```
var numbers = new List<int> { 1 };  
  
IEnumerable<int> query = numbers.Select (n => n * 10);    // Build query  
  
numbers.Add (2);                                         // Sneak in an extra element  
  
foreach (int n in query)  
    Console.Write (n + "|");                            // 10|20|
```



# Consulta em Arquivos e Database




- São usados para fazer a tradução das árvores de expressão para o código correspondente ao tipo da fonte de dados
- Principais tipos de providers
  - **LINQ to objects:** utilizado quando a fonte de consulta é um conjunto de objetos
  - **LINQ to XML:** utilizado quando a fonte de consulta é um XML
  - **LINQ to SQL:** utilizado quando a fonte de consulta é um banco de dados relacional
  - ...

# Exemplo de X-DOM



```
var bench = new XElement ("bench",  
    new XElement ("toolbox",  
        new XElement ("handtool", "Hammer"),  
        new XElement ("handtool", "Rasp")  
    ),  
    new XElement ("toolbox",  
        new XElement ("handtool", "Saw"),  
        new XElement ("powertool", "Nailgun")  
    ),  
    new XComment ("Be careful with the nailgun")  
);  
  
foreach (XNode node in bench.Nodes())  
    Console.WriteLine (node.ToString (SaveOptions.DisableFormatting) + ".");
```



```
<toolbox><handtool>Hammer</handtool><handtool>Rasp</handtool></toolbox>.  
<toolbox><handtool>Saw</handtool><powertool>Nailgun</powertool></toolbox>.  
<!--Be careful with the nailgun-->.
```



```
<toolbox><handtool>Hammer</handtool><handtool>Rasp</handtool></toolbox>.  
<toolbox><handtool>Saw</handtool><powertool>Nailgun</powertool></toolbox>.  
<!--Be careful with the nailgun-->.
```

```
IEnumerable<string> query =  
    from toolbox in bench.Elements()  
    where toolbox.Elements().Any (tool => tool.Value == "Nailgun")  
    select toolbox.Value;
```

```
RESULT: { "SawNailgun" }
```

```
int x = bench.Elements().Where (e => e.Name == "toolbox").Count(); // 2
```

```
int x = bench.Elements ("toolbox").Count(); // 2
```

# X-DOM e LINQ (2)



```
XElement contacts = XElement.Parse (  
@"<contacts>  
  <customer name='Mary' />  
  <customer name='Chris' archived='true' />  
  <supplier name='Susan'>  
    <phone archived='true'>012345678<!-- confidential --></phone>  
  </supplier>  
</contacts>");
```

```
contacts.Elements ("customer").Remove();
```

Remove todos os  
*customers*

```
contacts.Elements().Where (e => (bool?) e.Attribute ("archived") == true)  
  .Remove();
```

Remove  
*Chris*



- Precisa criar o *Object Model*
  - Representação do banco do dado em C#
- Várias maneiras de fazer
  - Usando o *Object Relational Designer* (Visual Studio)
  - Usando o *SQLMetal code-generation* tool
  - Usando T4 templates (com linq2db)
  - Usando qualquer editor (na mão)

```
using System;
using LinqToDB.Mapping;

[Table(Name = "Products")]
public class Product
{
    [PrimaryKey, Identity]
    public int ProductID { get; set; }

    [Column(Name = "ProductName"), NotNull]
    public string Name { get; set; }

    [Column]
    public int VendorID { get; set; }

    [Association(ThisKey = nameof(VendorID), OtherKey=nameof(Vendor.ID))]
    public Vendor Vendor { get; set; }

    // ... other columns ...
}
```



Tem várias maneiras de acessar uma DB usando LINQ

1. Entity Framework (EF Core)
2. LINQ to SQL
3. LinqToDB
  - Lighter and faster than the others
  - <https://linq2db.github.io/>

Vamos usar LinqToDb no lab!

```
public static List<Product> All()
{
    using (var db = new DbNorthwind())
    {
        var query = from p in db.Product
                     where p.ProductID > 25
                     orderby p.Name descending
                     select p;
        return query.ToList();
    }
}
```



The background features a network of gray lines connecting various colored circles (orange, yellow, blue, green) and a dark blue horizontal band with a subtle circuit pattern.

# Operadores LINQ

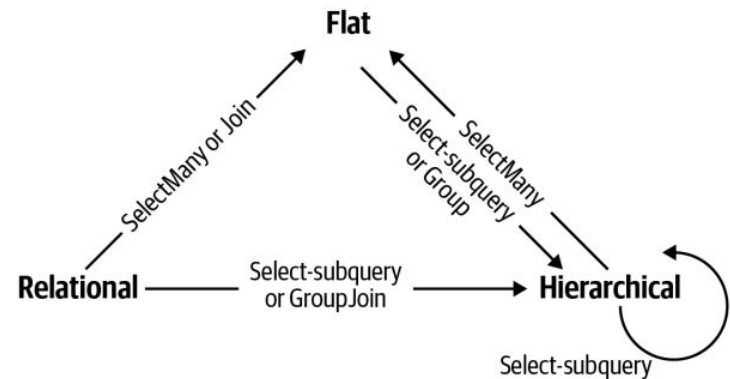


- Muita similaridade com os operadores de banco de dados
  - Usado em linguagem de consulta (SQL)
  - Alguns operadores não funciona com dados remotos
- 3 categorias
  - Sequência de entrada, sequência de saída (sequência → sequência)
  - Sequência de entrada, elemento único ou valor escalar de saída
  - Nada de entrada, sequência de saída (métodos de geração)

# Tipo de Operadores



- Filtração
- Projetação
- Juntar
- Ordenação
- Agrupamento
- Operadores de conjunto
- Métodos de conversão
- Operadores de Elementos
- Métodos de agregação
- Quantificadores
- Métodos de geração





## Retorna um subconjunto dos elementos originais

`IEnumerable<TSource> → IEnumerable<TSource>`

Method	Description	SQL equivalents
Where	Returns a subset of elements that satisfy a given condition	WHERE
Take	Returns the first <code>count</code> elements and discards the rest	WHERE ROW_NUMBER()... or TOP <code>n</code> subquery
Skip	Ignores the first <code>count</code> elements and returns the rest	WHERE ROW_NUMBER()... or NOT IN (SELECT TOP <code>n</code> ...)
TakeLast	Takes only the last <code>count</code> elements	Exception thrown
SkipLast	Takes only the last <code>count</code> elements	Exception thrown
TakeWhile	Emits elements from the input sequence until the predicate is false	Exception thrown
SkipWhile	Ignores elements from the input sequence until the predicate is false, and then emits the rest	Exception thrown
Distinct, DistinctBy	Returns a sequence that excludes duplicates	SELECT DISTINCT...



Transforma cada elemento com uma função lambda.  
SelectMany faz um flatten das sequências aninhadas

`IEnumerable<TSource> → IEnumerable<TResult>`

Method	Description	SQL equivalents
Select	Transforms each input element with the given lambda expression	SELECT
SelectMany	Transforms each input element, and then flattens and concatenates the resultant subsequences	INNER JOIN, LEFT OUTER JOIN, CROSS JOIN



- Faz a malha de elementos de uma sequência com outra
- O operador Zip enumera duas sequências aplicando uma função sobre cada par de elementos

`IEnumerable<TOuter>, IEnumerable<TInner> → IEnumerable<TResult>`

Method	Description	SQL equivalents
Join	Applies a lookup strategy to match elements from two collections, emitting a flat result set	INNER JOIN
GroupJoin	Similar to Join, but emits a <i>hierarchical</i> result set	INNER JOIN, LEFT OUTER JOIN
Zip	Enumerates two sequences in step (like a zipper), applying a function over each element pair	Exception thrown

`IEnumerable<TFirst>, IEnumerable<TSecond> → IEnumerable<TResult>`



Retorna uma reordenação de uma sequência

`IEnumerable<TSource> → IOrderedEnumerable<TSource>`

Method	Description	SQL equivalents
<code>OrderBy</code> , <code>ThenBy</code>	Sorts a sequence in ascending order	<code>ORDER BY ...</code>
<code>OrderByDescending</code> , <code>ThenByDescending</code>	Sorts a sequence in descending order	<code>ORDER BY ... DESC</code>
<code>Reverse</code>	Returns a sequence in reverse order	Exception thrown



Agrupar uma sequência em subsequências.

`IEnumerable<TSource> → IEnumerable<IGrouping<TKey, TElement>>`

Method	Description	SQL equivalents
GroupBy	Groups a sequence into subsequences	GROUP BY
Chunk	Groups a sequence into arrays of a fixed size	



# Operadores de conjunto



Recebe duas sequências do mesmo tipo e retorna uma nova sequência

`IEnumerable<TSource>, IEnumerable<TSource> → IEnumerable<TSource>`

Method	Description	SQL equivalents
Concat	Returns a concatenation of elements in each of the two sequences	UNION ALL
Union, UnionBy	Returns a concatenation of elements in each of the two sequences, excluding duplicates	UNION
Intersect, IntersectBy	Returns elements present in both sequences	WHERE ... IN (...)
Except, ExceptBy	Returns elements present in the first but not the second sequence	EXCEPT or WHERE ... NOT IN (...)



## Converte uma sequência

Method	Description
OfType	Converts IEnumerable to IEnumerable<T>, discarding wrongly typed elements
Cast	Converts IEnumerable to IEnumerable<T>, throwing an exception if there are any wrongly typed elements
ToArray	Converts IEnumerable<T> to T[]
ToList	Converts IEnumerable<T> to List<T>
ToDictionary	Converts IEnumerable<T> to Dictionary<TKey,TValue>
ToLookup	Converts IEnumerable<T> to ILookup<TKey,TElement>
AsEnumerable	Upcasts to IEnumerable<T>
AsQueryable	Casts or converts to IQueryable<T>



Seleciona um único elemento de uma sequência.

`IEnumerable<TSource> → TSource`

Method	Description	SQL equivalents
<code>First</code> , <code>FirstOrDefault</code>	Returns the first element in the sequence, optionally satisfying a predicate	<code>SELECT TOP 1 ... ORDER BY ...</code>
<code>Last</code> , <code>LastOrDefault</code>	Returns the last element in the sequence, optionally satisfying a predicate	<code>SELECT TOP 1 ... ORDER BY ... DESC</code>
<code>Single</code> , <code>SingleOrDefault</code>	Equivalent to <code>First/FirstOrDefault</code> , but throws an exception if there is more than one match	
<code>ElementAt</code> , <code>ElementAtOrDefault</code>	Returns the element at the specified position	Exception thrown
<code>MinBy</code> , <code>MaxBy</code>	Returns the element with the smallest or largest value	Exception thrown
<code>DefaultIfEmpty</code>	Returns a single-element sequence whose value is <code>default(TSource)</code> if the sequence has no elements	<code>OUTER JOIN</code>



- Executa um cálculo em uma sequência, retornando um valor escalar
  - normalmente um número

`IEnumerable<TSource> → scalar`

Method	Description	SQL equivalents
Count, LongCount	Returns the number of elements in the input sequence, optionally satisfying a predicate	COUNT ( ... )
Min, Max	Returns the smallest or largest element in the sequence	MIN ( ... ), MAX ( ... )
Sum, Average	Calculates a numeric sum or average over elements in the sequence	SUM ( ... ), AVG ( ... )
Aggregate	Performs a custom aggregation	Exception thrown



## Avalia um conjunto

`IEnumerable<TSource> → bool`

Method	Description	SQL equivalents
<code>Contains</code>	Returns <code>true</code> if the input sequence contains the given element	<code>WHERE ... IN (...)</code>
<code>Any</code>	Returns <code>true</code> if any elements satisfy the given predicate	<code>WHERE ... IN (...)</code>
<code>All</code>	Returns <code>true</code> if all elements satisfy the given predicate	<code>WHERE (...)</code>
<code>SequenceEqual</code>	Returns <code>true</code> if the second sequence has identical elements to the input sequence	



## Gera um novo conjunto

`void→IEnumerable<TResult>`

Method	Description
Empty	Creates an empty sequence
Repeat	Creates a sequence of repeating elements
Range	Creates a sequence of integers



Curso de Extensão  
Tecnologias Microsoft

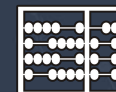


INF0992

Obrigado !!  
Merci !!

Hervé Yviquel  
hyviquel@unicamp.br

24 de Setembro de 2022



INSTITUTO DE  
COMPUTAÇÃO