UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Graduação em Ciência da Computação

Linguagens de Programação

Bruno Agrizzi
Gilberto Ewald
Luis Augusto
Nicholas Figueiredo

Histórico e Características

Em 1999 inicia-se o desenvolvimento de uma LP chamada Cool

Criada por Anders Hejlsberg e sua equipe

Em 2000 Cool passa a se chamar C#

A linguagem C# faz parte do conjunto de ferramentas oferecidas na plataforma .NET e surge como uma linguagem simples, robusta, orientada a objetos e fortemente tipada.

Anders Hejlsberg

Microsoft .Net Framework

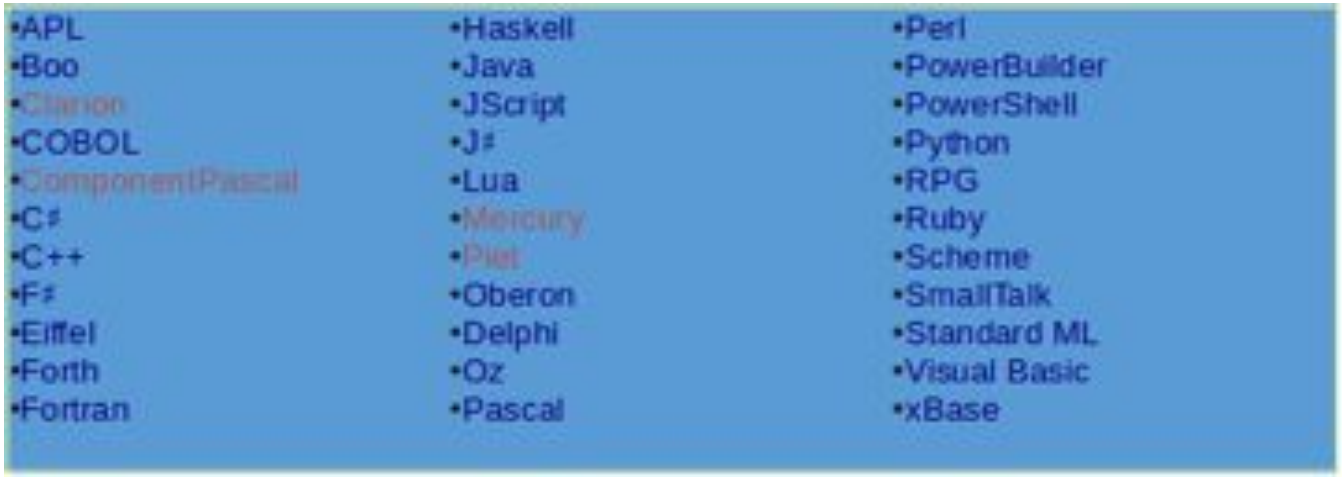
- **O que é Microsoft .Net Framework?**
  - É uma iniciativa da empresa Microsoft, que visa uma plataforma única para desenvolvimento e execução de sistemas e aplicações.
  - Todo e qualquer código gerado para **.NET** pode ser executado em qualquer dispositivo que possua um framework de tal plataforma.
  - Com idéia semelhante à plataforma Java, o programador deixa de escrever código para um sistema ou dispositivo específico, e passa a escrever para a plataforma **.NET**.

Microsoft .Net Framework

A plataforma **.NET** é executada sobre uma CommonLanguageRuntime - CLR (Ambiente de Execução Independente de Linguagem) interagindo com um Conjunto de Bibliotecas Unificadas (framework).

Esta CLR é capaz de executar, atualmente, mais de 33 diferentes linguagens de programação, interagindo entre si como se fossem uma única linguagem.

Microsoft .Net Framework



Download : msdn.microsoft.com/pt-br/netframework

Se atente a versões mais novas.

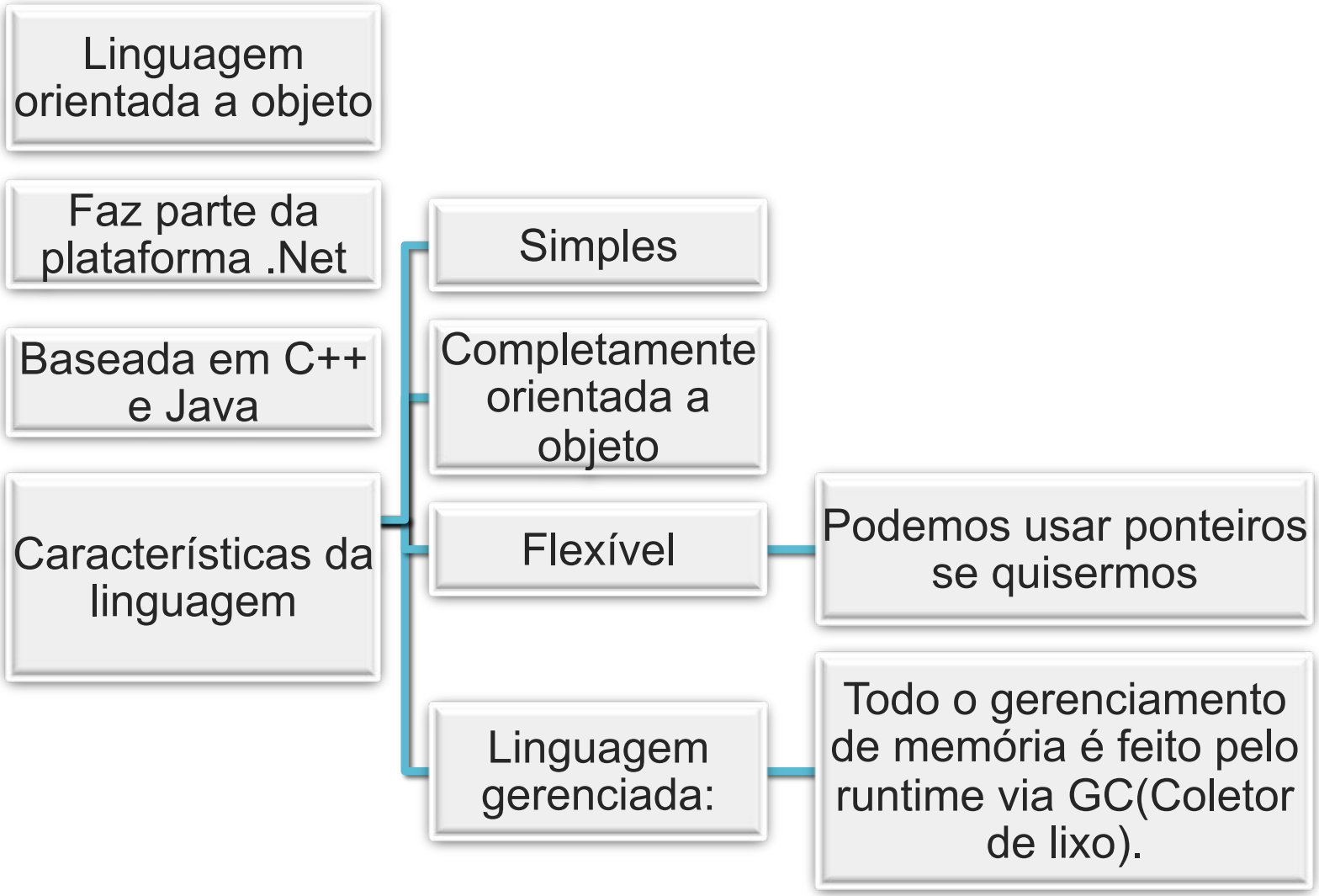
Microsoft Visual Studio

Conjunto de ferramentas criada para desenvolver em cima de uma plataforma .Net

Com ele fazemos sites, programas para Windows, dispositivos móveis.

Ferramenta única para vários tipos de aplicações.

C#


Linguagem orientada a objeto
Faz parte da plataforma .Net
Baseada em C++ e Java
Características da linguagem
Simples
Completamente orientada a objeto
Flexível
Podemos usar ponteiros se quisermos
Linguagem gerenciada:
Todo o gerenciamento de memória é feito pelo runtime via GC(Coletor de lixo).

Curiosidade

Alguns pensavam que C# simbolizava a sobreposição do simbolo +

O “#” vem do simbolo musical “sustenido”

O simbolo sustenido não tem no teclado portanto ficou “#”(cerquilha)

Compilador .NET

Algumas perguntas:

- Como funciona o processo de compilação/ interpretação de linguagens pertencentes a plataforma .NET(C#, VB e F#)?
- A linguagem C# é compilada, interpretada ou é modelo híbrido?

Relembrando:

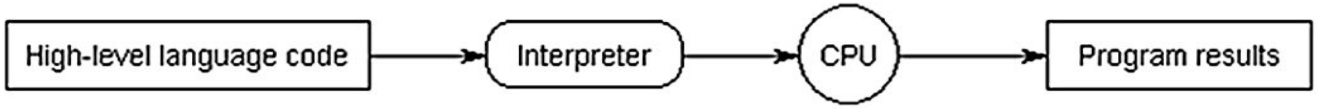
- É chamado de "**compilação**" o processo de transcrição de uma linguagem de alto nível para uma linguagem de baixo nível (também conhecida como linguagem de máquina).

Compilador .NET

Na **interpretação** a sequência de processamento é mais simplificada, e é realizada em três etapas:

Há uma diferença fundamental em relação ao processo de compilação: O tempo de execução.

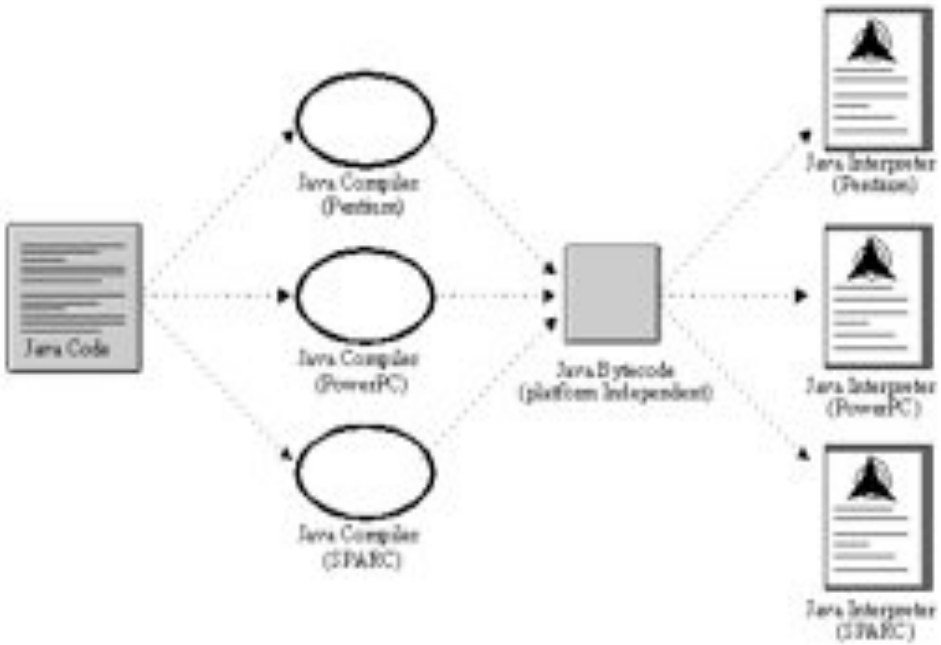
Neste processo, não existe qualquer geração de código intermediário ou etapas adicionais.



Compilador .NET

Modelo hibrido: Veio com a necessidade de sistemas que pudessem ser executados em qualquer computador independente da estrutura de hardware e de software disponíveis

Tentou unir vantagens de compilação tradicional (principalmente performance) e interpretação (principalmente a multiplataforma).



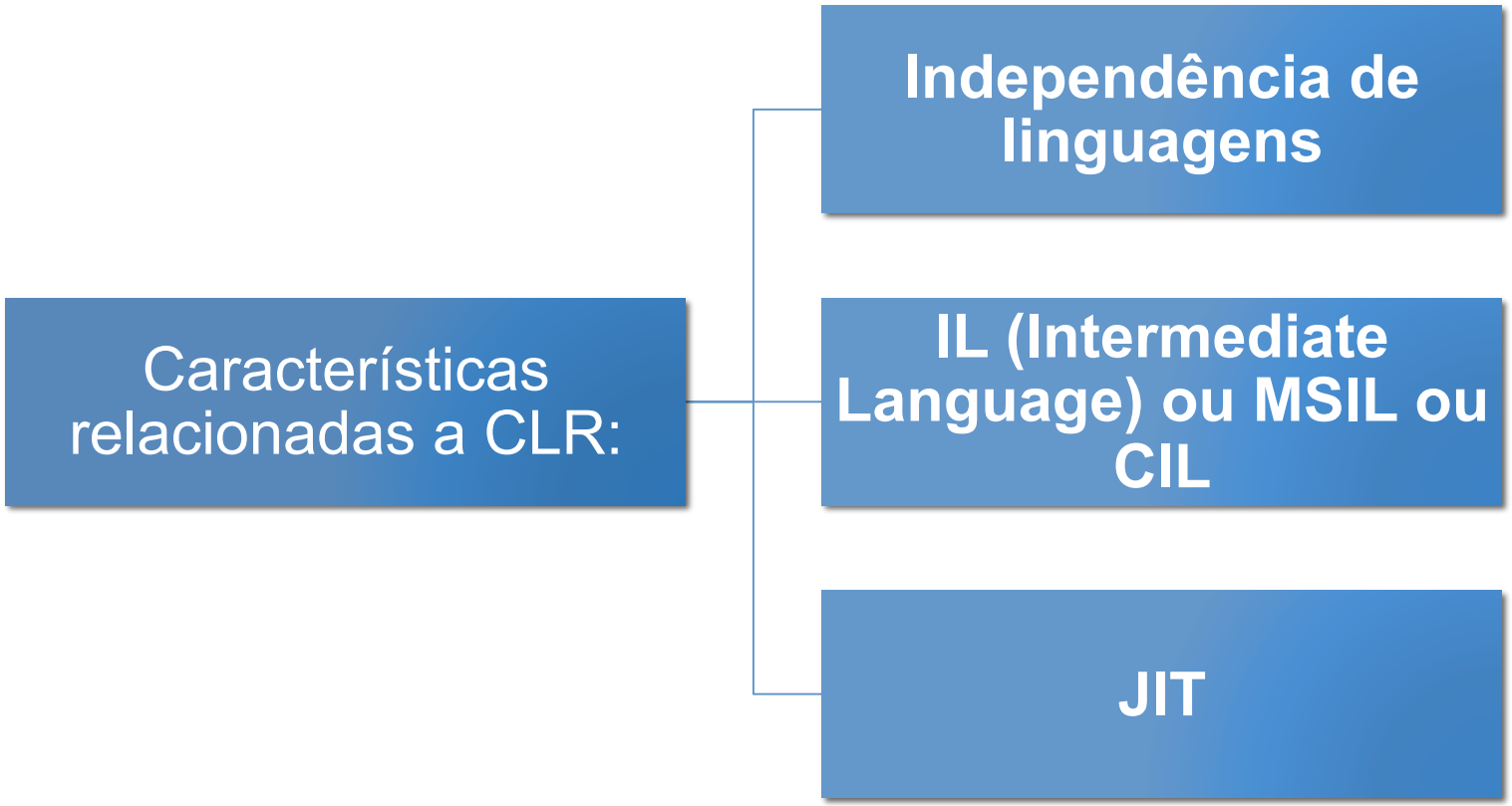
Compilador .NET

O processo de compilação/execução de aplicações da plataforma .NET é semelhante com o modelo da plataforma JAVA

A plataforma .NET disponibiliza um ambiente de execução, denominado Common Language Runtime

Para entender melhor , CLR está para .NET framework como a VM está para a plataforma Java

Compilador .NET



Compilador .NET

Independência de linguagens

- Na plataforma .NET não se está preso a uma linguagem específica, sendo possível em um mesmo ambiente de desenvolvimento, encontrar trechos de uma aplicação escritas em Visual Basic (por exemplo) e trechos escritos em C# e, ainda assim, a aplicação funcionar perfeitamente bem.
- Isto somente é possível graças a IL (ou MSIL, falaremos dela mais adiante) e a interpretação dela por parte da CLR.
- Dois Aspectos fundamentais para que ocorra a interoperabilidade:
  - Common Type-System (CTS) é responsável por definir os tipos de dados suportados pela plataforma
  - Common Language Specification (CLS). define os requisitos mínimos necessários para que uma linguagem possa funcionar dentro da .NET framework.

Compilador .NET

IL (Intermediate Language) ou MSIL ou CIL

- No .NET, temos a linguagem intermediária (IL) que serve de base para execução da aplicação dentro da CLR

JIT

- Responsável por otimizar o processo de geração de IL.
- Imagine por exemplo que, alguns objetos foram instanciados no decorrer de seu código mas nunca foram utilizados. Neste caso, p JIT entraria em ação e não geraria código IL para algo que nunca será utilizado.

http://fabriciosanchez.com.br/2/teste-de-performance-net-versus-java-pipeline-de-compilacaointerpretacao/

Compilador .NET

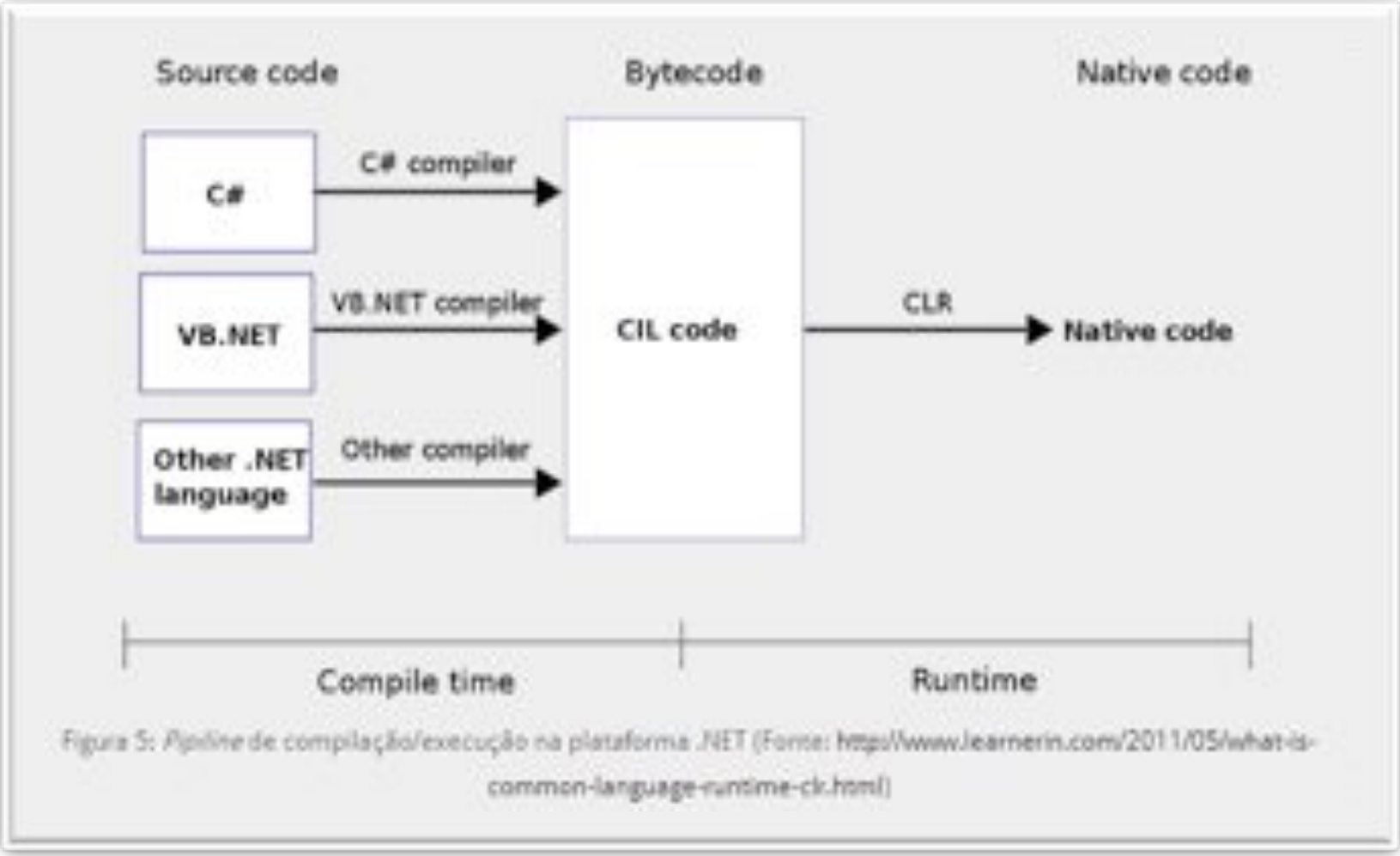


Figura 5: *Pipeline* de compilação/execução na plataforma .NET (Fonte: http://www.learnerin.com/2011/05/what-is-common-language-runtime-clr.html)

Compilador .NET

Caso queira aprender mais como IL funciona, recomendamos esse post: http://elemarjr.net/2010/07/28/il-101parte-1/

Comparação da estrutura de JAVA com C#

Package x Namespace

JAVA	C#

• Package
• Um pacote representa fisicamente uma pasta
	• Namespace
	• Namespace não está relacionado a pasta
	  • É possível ter pasta com um nome e classes que pertençam a um namespace com outro nome
	  • Ou ainda, é possível dentro de uma pasta existirem classes com namespaces diferentes

```
package pojo;
```
Pacote

```
namespace Poco
```
Namespace da a classe

Comparação da estrutura de JAVA com C#

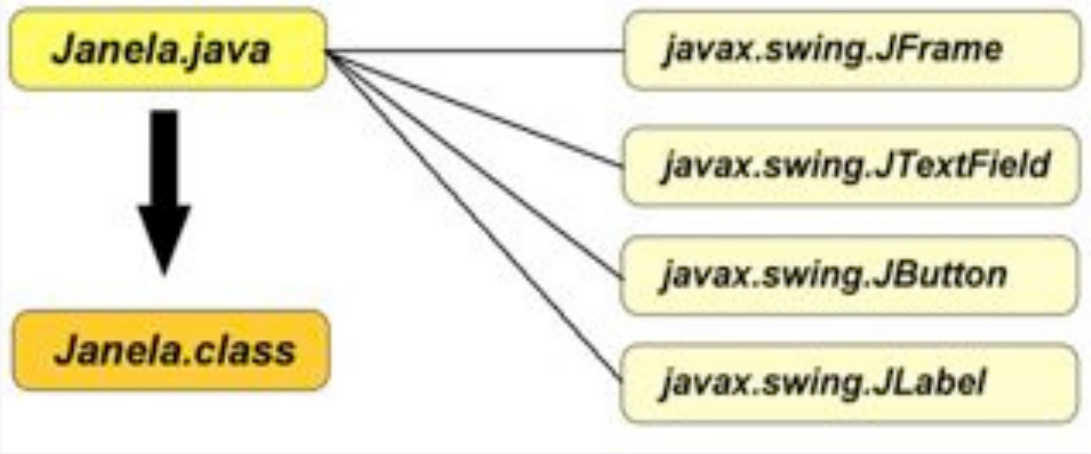
Importação de classes utilizadas na implementação

JAVA	C#
• Bibliotecas são importadas com import, e situa-se abaixo da declaração do pacote	• As classes são importadas através da instrução using, e se encontra antes da declaração do namespace, pois em C# a classe é delimitada pelo namespace a que pertence.z

Comparação da estrutura de JAVA com C#

Importação de classes utilizadas na implementação

Funcionamento do processo de compilação e geração de bytecodes.



Situação onde a classe Janela.java faz uso das classes Jframe,JtextField,Jbutton,JLabel

Comparação da estrutura de JAVA com C#

Importação de classes utilizadas na implementação

JAVA	C#
• Após a compilação, embora a classe use outras quatro, apenas um arquivo é gerado Janela.class	• Em c# ao invés de bytecodes, são gerados assemblies • Os assemblies gerados podem ser aplicações(exe) ou bibliotecas(dll)

Comparação da estrutura de JAVA com C#

Declaração de classes

JAVA	C#
• Não há o conceito de **partial class** (classes parciais)	• Uma característica oferecida não tem em JAVA é o conceito de **partial class** (classes parciais) • É uma classe que possui sua implementação distribuída em mais de um arquivo. • Facilmente visualizado em aplicações **Windows Form**, onde a interface(formulário) esta em uma arquivo e o comportamento em outro

```
public class Cliente {
```


```
public class Cliente {
```

Comparação da estrutura de JAVA com C#

Modificadores de acesso

C#

- Em C# além dos conhecidos modificadores de acesso – public, private e protected, existem ainda os **internal** e **protected internal**

Comparação da estrutura de JAVA com C#

Atributos e Propriedades

JAVA	C#
• Um atributo é declarado propriedade quando o mesmo possui métodos assessores públicos declarados para ele, ou seja, os métodos **get** e **set**.	• Segue o padrão Delphi, ou seja, os valores são obtidos e atribuídos através de um operador de igualdade(**=**). • Perceba na imagem a seguir que o get e set de C# é diferente.

Comparação da estrutura de JAVA com C#

Atributos e Propriedades

```
private long id;
private String nome;
private Calendar nascimento;
private double renda;

public long getId() {
    return id;
}

public void setId(long id) {
    this.id = id;
}
```

Atributos

Acessores

```
private long _id;

public long Id
{
    get { return _id; }
    set { _id = value; }
}
```

Atributo

Acessores

Comparação da estrutura de JAVA com C#

Atributos e Propriedades

- Utilizando os assessores:


```
Cliente cliente = new Cliente();

// Atribuição
cliente.setNome("Everton");

// Recuperação
String nome = cliente.getNome();
```

```
Cliente cliente = new Cliente();

// Atribuição
cliente.Nome = "Everton";

// Recuperação
string nome = cliente.Nome;
```

Comparação da estrutura de JAVA com C#

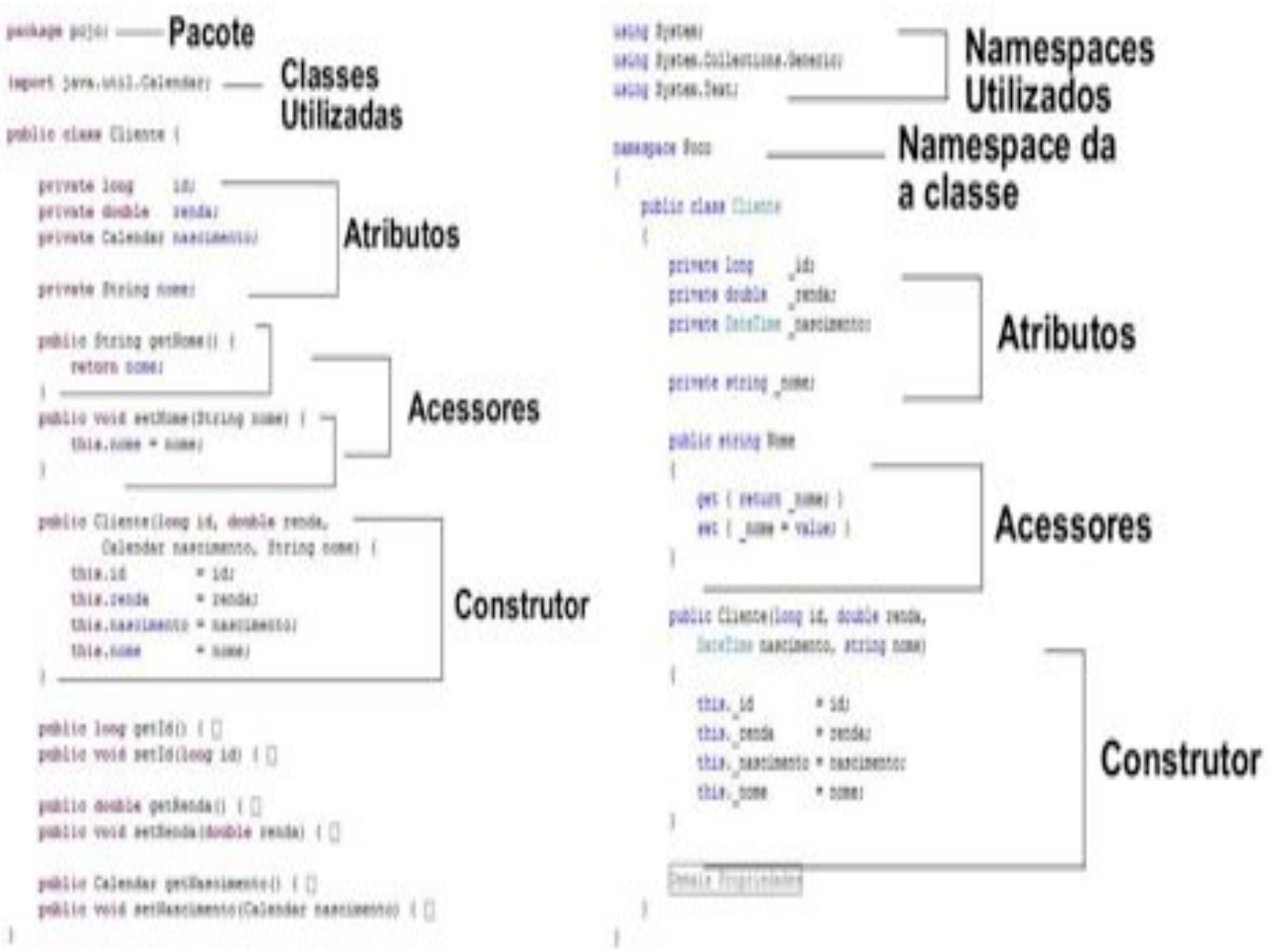
Construtores

```java
public Cliente(long id, String nome,
        Calendar nascimento, double renda) {
    this.id = id;
    this.nome = nome;
    this.nascimento = nascimento;
    this.renda = renda;
}
```

Construtor

```
public Cliente(long id, string nome,
        DateTime nascimento, double renda)
{
    this._id = id;
    this._nome = nome;
    this._nascimento = nascimento;
    this._renda = renda;
}
```

Construtor


Pacote
Classes Utilizadas
Atributos
Acessores
Construtor
Namespaces Utilizados
Namespace da a classe
Atributos
Acessores
Construtor

Estrutura das classes usadas no Trabalho

- *Entregador*

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

namespace DisqueRango
{
    class Entregador
    {
        private string nome;
        private bool presente;
        private string placa;
        private int entregas;
        private double comissao;
        private List<Pedido> listaPedidos;


        public Entregador(string nome,string placa)
        {
            this.nome = nome;
            this.placa = placa;
        }

        public string Nome
        {
            get { return nome; }
            set { nome = value; }
        }
```

Estrutura das classes usadas no Trabalho

- *Pedido*

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

namespace DisqueRango
{
    class Pedido
    {
        private Cliente cliente;
        private List<ItemPedido> itensPedidos;
        private Entregador entregador;
        private DateTime data;
        private DateTime hora;

        internal Cliente Cliente
        {
            get { return cliente; }
            set { cliente = value; }
        }

        public DateTime Data
        {
            get { return data; }
            set { data = value; }
        }

        public DateTime Hora
        {
            get { return hora; }
            set { hora = value; }
        }
```

Variáveis

O nome do tipo das variáveis são bem parecidas com as que estamos acostumados a ouvir:

- int denota inteiro
- string denota string
- bool = boolean
- double = double

- O exemplo a seguir exemplifica o uso de variáveis em C#
- Note que **Console.WriteLine(String)**, é uma função que imprime na tela o valor passado como parâmetro
- No final do programa devemos chamar a função **Console.ReadyKey()**, para que ao ser executado o programa não feche imediatamente.

Variáveis

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

namespace Variaveis
{
    class Program
    {
        static void Main(string[] args)
        {
            //uso de variaveis
            int idade = 30; //inteiro
            string nome = "Maria Carvalho"; // string
            bool solteira = true; // booleano
            double salario = 2300.50; //double

            Console.WriteLine("Nome = " + nome);//imprime na tela
            Console.WriteLine("idade = " + idade);
            Console.WriteLine("salario = " + salario);
            Console.WriteLine("solteira = " + solteira);

            Console.ReadKey(); //Faz o programa parar e só continuar quando apertar uma tecla


        }
    }
}
```

Variável ponteiro

Em C# também podemos utilizar ponteiros

É declarado do seguinte modo:

- type* identifier;

Quando você declara vários ponteiros na mesma declaração, o * é gravado em conjunto com o tipo subjacente apenas, não como um prefixo de nome de cada ponteiro, exemplo:

```
int* p1, p2, p3;   // Ok
```

```
int *p1, *p2, *p3;   // Invalido em C#
```

Variável ponteiro

Exemplo	Descrição
`int* p`	p é um ponteiro para um inteiro
`int** p`	p é um ponteiro para um ponteiro para um inteiro
`int*[] p`	p é uma matriz unidimensional de ponteiros para inteiros
`char* p`	p é um ponteiro para um char.
`void* p`	p é um ponteiro para um tipo desconhecido

Variável ponteiro

Stackalloc:

- A palavra-chave é usada para alocar um bloco de memória na pilha.
- Exemplo:
  - int* block = stackalloc int[100];

O exemplo a seguir, calcula e exibe os 20 primeiros números da sequência de Fibonacci.

- Obs: Foi botado no output até o 8 por questões de ajuste no slide.

a palavra chave **unsafe** denota um contexto sem segurança, o que é necessário para qualquer operação que envolva ponteiros.

```
class Test
{
    static unsafe void Main()
    {
        const int arraySize = 20;
        int* fib = stackalloc int[arraySize];
        int* p = fib;
        // The sequence begins with 1, 1.
        *p++ = *p++ = 1;
        for (int i = 2; i < arraySize; ++i, ++p)
        {
            // Sum the previous two numbers.
            *p = p[-1] + p[-2];
        }
        for (int i = 0; i < arraySize; ++i)
        {
            Console.WriteLine(fib[i]);
        }

        // Keep the console window open in debug mode.
        System.Console.WriteLine("Press any key to exit.");
        System.Console.ReadKey();
    }
}
/*
Output
1
1
2
3
5
8
```

Obtendo valor de uma variável do tipo ponteiro

- No exemplo a seguir, uma variável do tipo char é acessada através de ponteiros de tipos diferentes
- Observe que o endereço do theChar irá variar de execução em execução, como o endereço físico alocado a uma variável pode ser alterado.
- Note que na função Console.WriteLine, temos o especificador do formato,ex: {0:x2}(hexadecimal) (mostra em que formato será impresso o valor)

Mais informações: http://msdn.microsoft.com/en-us/library/s8s7t687.aspx

Obtendo valor de uma variável do tipo ponteiro

```
unsafe class TestClass
{
    static void Main()
    {
        char theChar = 'Z';
        char* pChar = &theChar;
        void* pVoid = pChar;
        int* pInt = (int*)pVoid;

        System.Console.WriteLine("Value of theChar = {0}", theChar);
        System.Console.WriteLine("Address of theChar = {0:X2}",(int)pChar);
        System.Console.WriteLine("Value of pChar = {0}", *pChar);
        System.Console.WriteLine("Value of pInt = {0}", *pInt);
    }
}
```

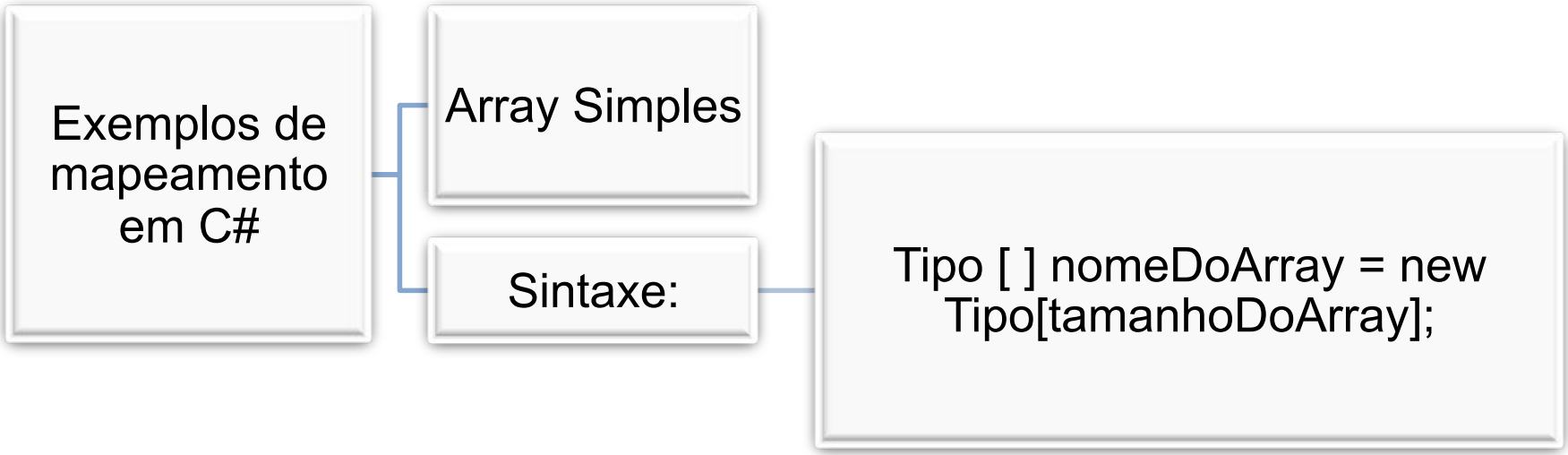
```
              Valor de theChar = Z
Endereço do theChar = 12F718
Valor de pChar = Z
Valor de pInt = 90
```

Produto Cartesiano

- Em C# temos structs
- Um tipo struct, é um tipo de valor normalmente usado para encapsular pequenos grupos de variáveis relacionadas.

```
public struct Book
{
    public decimal price;
    public string title;
    public string author;
}
```

Mapeamento


Exemplos de mapeamento em C#
Array Simples
Sintaxe:
Tipo [ ] nomeDoArray = new Tipo[tamanhoDoArray];

Mapeamento

- *Exemplo*

```
static void Main(string[] args)
{
    //sintaxe = tipo [] nome_do_array = new tipo[tamanho_do_array];
    //double[] notas = new double[5];
    //notas[0] = 6;
    //notas[1] = 8;
    //notas[2] = 7;
    //notas[3] = 9;
    //notas[4] = 5;
    //double soma = notas[0] + notas[1] + notas[2] + notas[3] + notas[4];
    //double media = soma / 5;

    double[] notas = {6,8,7,9,5};

    double soma = notas.Sum(); //Método que calcula a soma
    double media = notas.Average();//Método que calcula a média

    Console.WriteLine("A soma das notas é "+soma);
    Console.WriteLine("A media das notas é "+media);

    Console.ReadKey();
}
```

Mapeamentos

Mapeamentos

- *Exemplo*

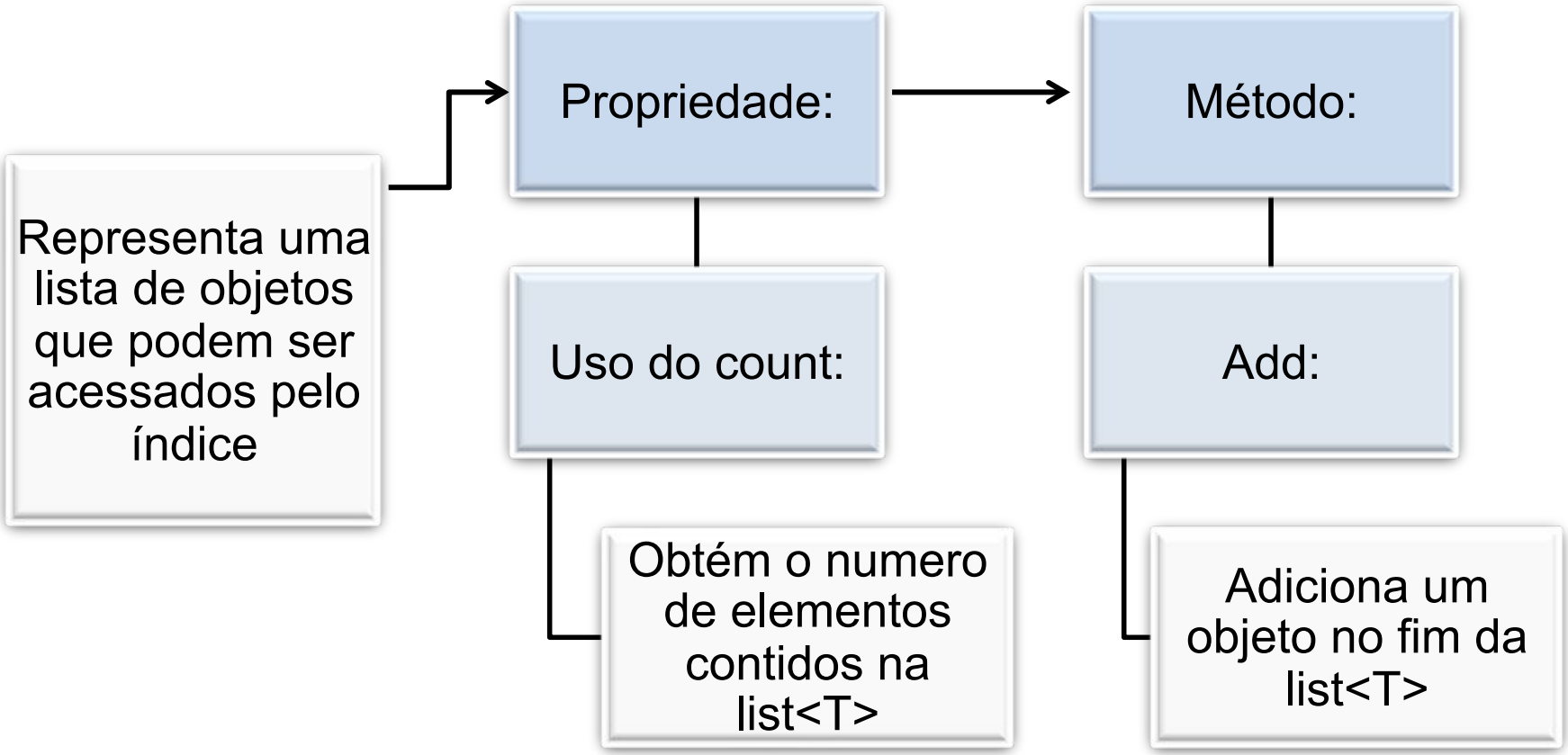
```
static void Main(string[] args)
{
    double[,] aluno_nota = new double[3, 3];
    aluno_nota[0, 0] = 8;
    aluno_nota[0, 1] = 6;
    aluno_nota[0, 2] = 4;
    aluno_nota[1, 0] = 4;
    aluno_nota[1, 1] = 7;
    aluno_nota[1, 2] = 9;
    aluno_nota[2, 0] = 3;
    aluno_nota[2, 1] = 8;
    aluno_nota[2, 2] = 1;
    for (int aluno = 0; aluno < 3; aluno++)
        for (int nota = 0; nota < 3; nota++)
            Console.WriteLine("Array Matriz aluno_nota[" + aluno + "," + nota + "] = " + aluno_nota[aluno,nota]);
    Console.ReadKey();


}
```

Operadores Aritméticos

Os operadores aritméticos seguem a mesma ideia das Lps já estudadas anteriormente:

```
static void Main(string[] args)
{
    double num1, num2;
    num1 = 10;
    num2 = 5;
    double soma = num1 + num2;
    double multiplicacao = num1 * num2;
    double divisao = num1 / num2;
    double subtracao = num1 - num2;
    double media = soma / 2;

    Console.WriteLine("A soma de " + num1 + " e "+num2+" = "+soma);
    Console.WriteLine("A multiplicacao de " + num1 + " e " + num2 + " = " + multiplicacao);
    Console.WriteLine("A divisao de " + num1 + " e " + num2 + " = " + divisao);
    Console.WriteLine("A subtracao de " + num1 + " e " + num2 + " = " + subtracao);
    Console.WriteLine("A media de " + num1 + " e " + num2 + " = " + media);
    Console.ReadKey();
}
```

Classe List<T>



Uso da classe List<T> no trabalho

Alguns métodos da classe **DiskRango** que ilustram o uso de count e add.

```
public void adicionarEntregador(Entregador entregador)
{
    this.listaEntregadores.Add(entregador);
}

public void imprimirEntregadores()
{
    for (int i = 0; i < this.listaEntregadores.Count; i++)
    {
        Console.WriteLine(listaEntregadores[i].Nome);
        Console.WriteLine(listaEntregadores[i].Placa);
    }
}
```

Uso de variáveis no trabalho

- *Classe Cliente*

```
class Cliente
{
    private string nome;
    private string endereco;
    private string telefone;
    private string pontoReferencia;
    private List<Pedido> listaPedidos;
```

- *Classe Refeição*

```
class Refeicao : ItemCardapio
{
    bool quente;
```

Uso de variáveis no trabalho

- *Classe Bebida*

```
class Bebida : ItemCardapio
{

    double volume;
```

- *Classe Cardápio*

```
public class Cardapio
{
    private List<ItemCardapio> itensCardapio = new List<ItemCardapio>();
```

Uso de variáveis no trabalho

- *Classe Entregador*

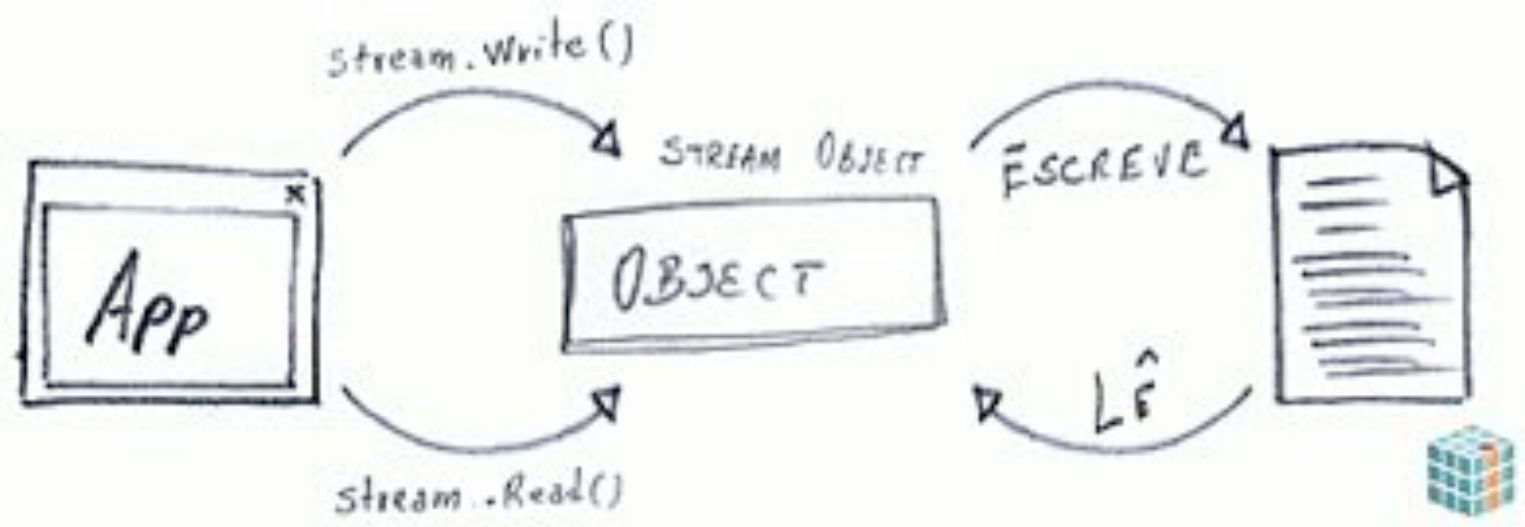
```
class Entregador
{
    private string nome;
    private bool presente;
    private string placa;
    private int entregas;
    private double comissao;
    private List<Pedido> listaPedidos;
```

- *Classe ItemPedido*

```
class ItemPedido
{
    private ItemCardapio itemCardapio;
    private int quantidade;
```

Leitura e escrita em arquivos

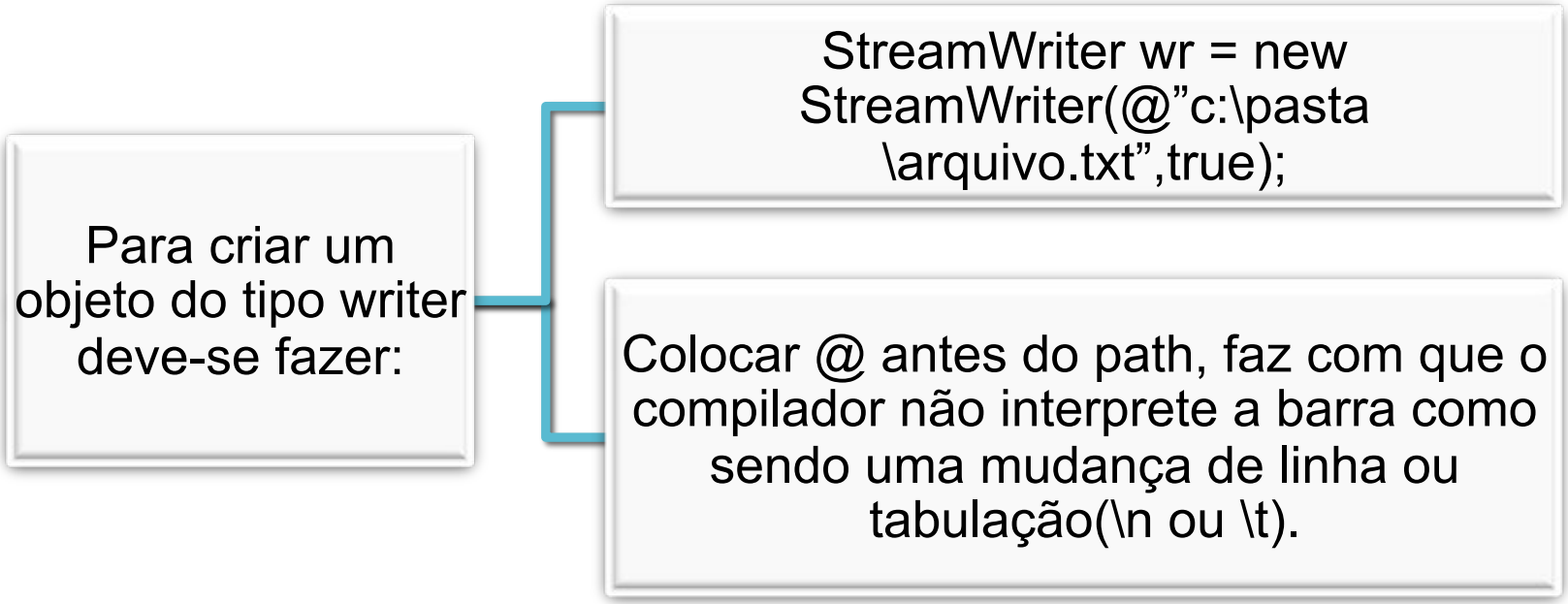
- C# utiliza Stream para ler e escrever em arquivos
- Sempre que pretender ler ou escrever dados(bytes) num arquivo, deve ser utilizado ou criado um objeto Stream.



Leitura e escrita em arquivos

A classe Stream está disponível na biblioteca IO, incluindo no arquivo da classe:

- Using System.IO;



Leitura e escrita em arquivos

A seguir podemos escrever no arquivo, a função de escrita:

wr.WriteLine(“Este será escrito no arquivo”);

Fechando o arquivo:

wr.close()

Leitura e escrita em arquivos

Para leitura de arquivos utiliza-se o StreamReader:

```
StreamReader rd = new StreamReader(@”c:\pasta\ficheiro.txt”);
```

No exemplo a seguir vamos criar um arquivo onde se escreverá duas linhas e, depois de escrito, iremos ler essas duas linhas.

Leitura e escrita em arquivos

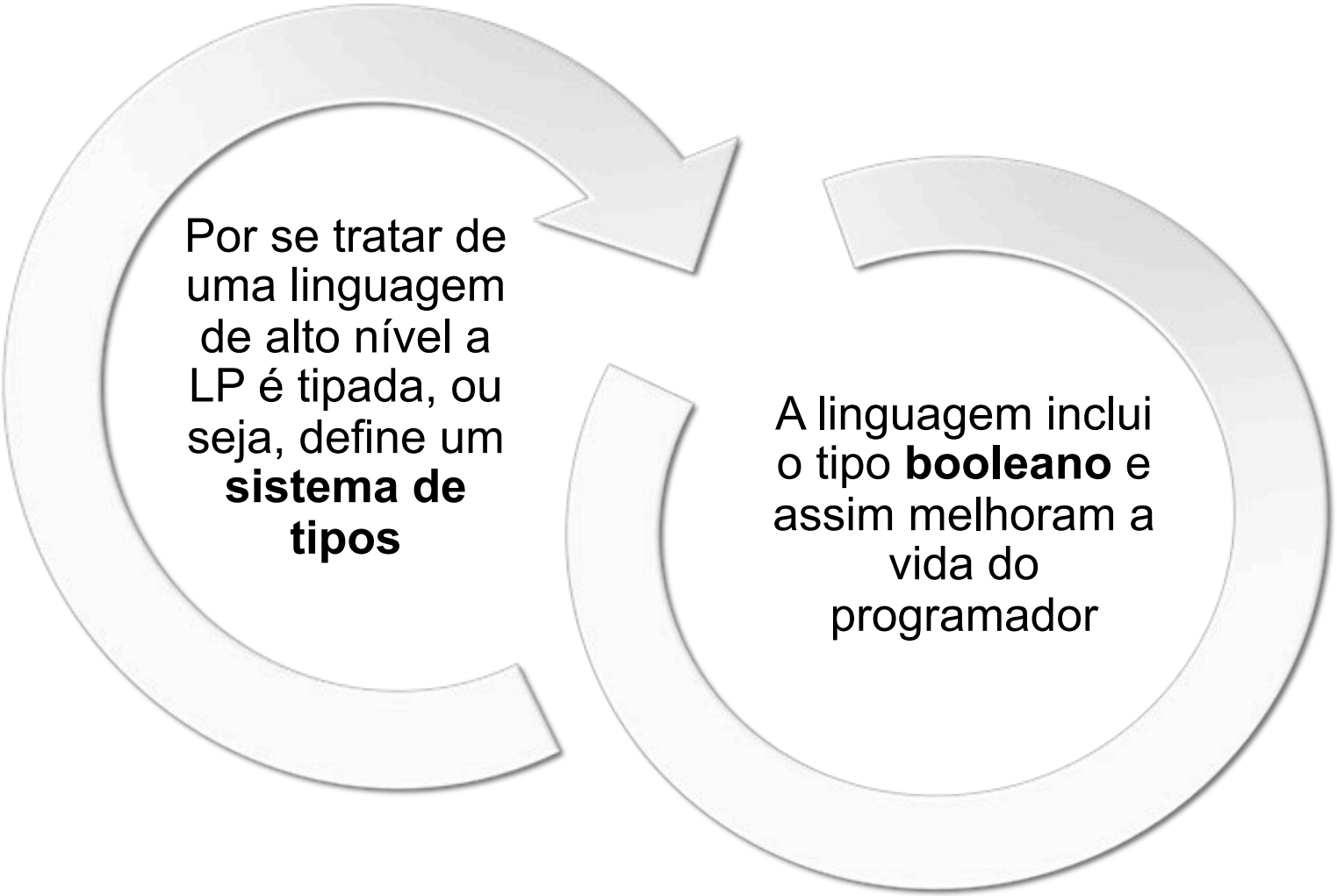
```
StreamWriter wr = new StreamWriter(@"c:\pasta\doc.txt", true);
wr.WriteLine("Primeira linha");
wr.WriteLine("Segunda linha");
wr.Close();
StreamReader rd = new StreamReader(@"c:\pasta\doc.txt");
while (!rd.EndOfStream)
{
    string linha = rd.ReadLine();
    Console.WriteLine(linha);
}
rd.Close();
```

Leitura e escrita em arquivos

- *Exemplo do Trabalho*

```
namespace TrabLp
{
    class Arquivo
    {
        public DiskRango lerArquivo(string pathEntregador,string pathItensCardapio)
        {
            DiskRango diskRango = new DiskRango();
        try
        {
            StreamReader sr = new StreamReader(pathEntregador);

                int count = 0;
                while (sr.Peek() >= 0)
                {

                    //Lendo a linha
                    string[] linha = sr.ReadLine().Split(';');

                    //Vai servir mesmo só da segunda linha em diante
                    if (count != 0)
                    {

                        //Criando o objeto entregador
                        Entregador entregador = new Entregador(linha[0],linha[1]);

                        //Adicionando a lista de entregadores do diskRango
                        diskRango.adicionarEntregador(entregador);
                    }
                    count++;
                }
                sr.Close();
```

Polimorfismo



Sistemas de tipos

- C# faz verificação de tipos misto mista
  - Maior parte das verificações de tipos em tempo de compilação
  - Outras em tempo de execução

- C# é uma LP quase fortemente tipada
  - Algumas operações não são permitidas, exemplo somar int com bool.

Sistemas de tipos

- C# não permite

```
static void Main(string[] args)
{
    int a = 3;
    bool c = true;
    Console.WriteLine(c + a);
    Console.ReadKey();
}
```

- Por outro lado podemos somar int com double, e até int com char, que não teremos problemas ao executar

Sistemas de tipos

- C# permite

```
static void Main(string[] args)
{
    int a = 3;
    double b = 4.5;
    Console.WriteLine(a+b);

    int c = 5;
    char d = 'g';
    Console.WriteLine(c + d);

    Console.ReadKey();
}
```

Conversão implícita de tipos

- Situações nas quais se aplica um operando de tipo diferente do esperado por uma operação
  - C# adota uma postura intermediaria
- Conversão de tipos baseadas no conceito de inclusão de tipos.

```
static void Main(string[] args)
{
    int a = 3;
    int b = 4;
    double c = a + b;
    Console.WriteLine(c);
    Console.ReadKey();
}
```

Conversão implícita de tipos

- Erro ao tentarmos executar o código:

```
static void Main(string[] args)
{
    double a = 3;
    double b = 4;
    int c = a + b;
    Console.WriteLine(c);
    Console.ReadKey();
}
```

- Necessário o uso do cast.

```
static void Main(string[] args)
{
    double a = 3;
    double b = 4;
    int c = (int)(a + b);
    Console.WriteLine(c);
    Console.ReadKey();
}
```

Polimorfismo

Ad-hoc

- Sobrecarga:
  - Vários métodos com o mesmo nome
  - No código exemplo a seguir existem vários métodos calcular
  - A função calcular correta é chamada de acordo com os parâmetros que recebe
  - A saída nos mostra qual função de fato foi chamada a medida que invocamos o método calcular.

```
class Program
{
    static double media;
    static string metedo_sobrecarga;

    public static void calcular(double nota1, double nota2)
    {
        media = (nota1 + nota2) / 2;
        metedo_sobrecarga = "calcular(double nota1, double nota2)";
    }

    public static void calcular(string nota1, string nota2)
    {
        media = (double.Parse(nota1) + double.Parse(nota2)) / 2;
        metedo_sobrecarga = "calcular(string nota1, string nota2)";
    }

    public static void calcular(double nota1, string nota2)
    {
        media = (nota1 + double.Parse(nota2)) / 2;
        metedo_sobrecarga = "calcular(double nota1, string nota2)";
    }
```

```
    public static void mostrar_media()
    {
        Console.WriteLine("A media é " + media);
        Console.WriteLine("Método chamado " + metodo_sobrecarga+"\n");
        Console.ReadKey();
    }

    static void Main(string[] args)
    {
        calcular(5, 5);
        mostrar_media();

        calcular("5", "5");
        mostrar_media();

        calcular(5, "5");
        mostrar_media();


    }
}
```

```
A media é 5
Método chamado calcular(double nota1, double nota2)

A media é 5
Método chamado calcular(string nota1, string nota2)

A media é 5
Método chamado calcular(double nota1, string nota2)
```

Polimorfismo

Universal

- Paramétrico:
  - Genérics foram adicionados à versão 2.0 da linguagem C#
  - Introduzem o conceito de parâmetros de tipos, que tornam possíveis a estruturação de classes e métodos que adiam a especificação de um ou mais tipos até que a classe ou método seja declarada e instanciada pelo código do cliente.
  - Podemos escrever uma única classe que outro código do cliente poderá usar sem aumentar o custo ou risco de conversões (cast) em tempo de execução (runtime)

```
// Declara a classe genérica
public class GenericList<T>
{
    void Add(T input) { }
}
class TestGenericList
{
    private class ExampleClass { }
    static void Main()
    {
        // Declara uma lista do tipo int
        GenericList<int> list1 = new GenericList<int>();

        // Declara uma lista do tipo string
        GenericList<string> list2 = new GenericList<string>();

        // Declara uma lista do tipo ExampleClass.
        GenericList<ExampleClass> list3 = new GenericList<ExampleClass>();
    }
}
```

Genérico

- Métodos genéricos
  - É um método que é declarado com parâmetros de tipo da seguinte forma:

```
static void Swap<T>(ref T lhs, ref T rhs)
{
    T temp;
    temp = lhs;
    lhs = rhs;
    rhs = temp;
}
```

  - Exemplo a seguir mostra uma maneira de chamar o método usando int para argumento de tipo

```
public static void TestSwap()
{
    int a = 1;
    int b = 2;

    Swap<int>(ref a, ref b);
    System.Console.WriteLine(a + " " + b);
}
```

Genérico

- Podemos também omitir o tipo no argumento, e o compilador inferirá, a seguinte chamada para swap, é equivalente à anterior.

```
Swap(ref a, ref b);
```

- Em uma classe genérica, métodos não genéricos podem acessar os parâmetros de tipo de nível de classe, da seguinte maneira

```
class SampleClass<T>
{
    void Swap(ref T lhs, ref T rhs) { }
}
```

Genérico

- Se definirmos métodos genéricos com os mesmos parâmetros de tipos da classe contingente, o compilador gera **CS0693** de aviso.
  - Motivo: Dentro do escopo do método fornecido para o T interior, oculta o argumento fornecido para o T exterior

- Se queremos flexibilidade em trabalhar com métodos genéricos com argumentos diferentes dos fornecidos pela classe instanciada, devemos fornecer outro identificador para o parâmetro de tipo do método, como mostra o exemplo a seguir.

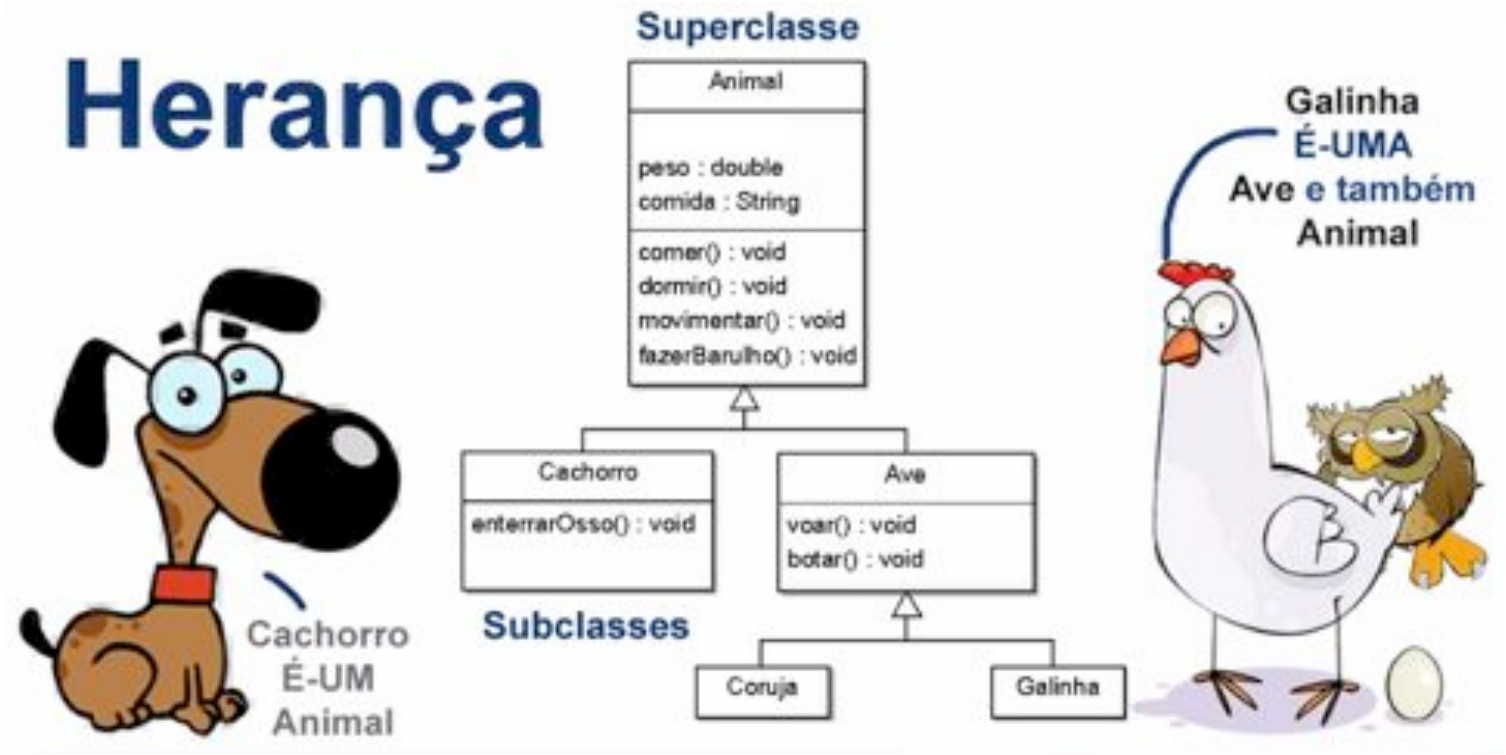
Genérico

- *Exemplo*

```
class GenericList<T>
{
    // CS0693
    void SampleMethod<T>() { }
}

class GenericList2<T>
{
    //No warning
    void SampleMethod<U>() { }
}
```

Polimorfismo

Inclusão



Polimorfismo

Herança

- Para codificar que uma classe estende da outra, usamos a notação “classeA : classeB”
- Como fazer para o construtor chame explicitamente o construtor da superclasse?
  - Basta fazer uso da notação: “'modificadorDeAcesso' subclasse”(parametros):base(parametros)

Polimorfismo

- *Exemplo*

```
namespace ExemploHeranca
{
    class Animal
    {
        private double peso;
        private string comida;

        public Animal(double peso, string comida)
        {
            this.peso = peso;
            this.comida = comida;
        }

        public string Comida
        {
            get { return comida; }
            set { comida = value; }
        }

        public double Peso
        {
            get { return peso; }
            set { peso = value; }
        }

        public void dormir() { Console.WriteLine("Dormiu"); }
        public void comer() { Console.WriteLine("Comeu"); }
        public void fazerBarulho() { Console.WriteLine("Fez Barulho"); }
        public void alimentar() { Console.WriteLine("Alimentou"); }
```

```
namespace ExemploHeranca
{
    class Cachorro : Animal
    {
        private bool bravo;

        public Cachorro(bool bravo,double peso,string comida) : base(peso,comida)
        {
            this.bravo = bravo;
        }
    }
}
```

```
namespace ExemploHeranca
{
    class Galinha : Animal
    {
        private bool cantaCedo;

        public Galinha(bool cantaCedo,double peso,string comida) : base(peso,comida)
        {
            this.cantaCedo = cantaCedo;
        }
    }
}
```

```
namespace ExemploHeranca
{
    class AnimalTest
    {
        static void Main(string[] args)
        {
            Cachorro viraLata = new Cachorro(true,30.5,"carne");

            viraLata.dormir();

            Galinha cocorico = new Galinha(false,4.3,"milho");
            cocorico.dormir();


            if (viraLata is Animal)
            {
                Console.WriteLine("ViraLata eh um animal");
            }

            Console.ReadKey();
        }
    }
}
```

```
Dormiu
Dormiu
ViraLata eh um animal
```

Polimorfismo

Sobrescrita de métodos

- Na superclasse devemos ter o modificador de acesso **virtual**
- Virtual indica que o método pode ser sobrescrito na classe derivada
- O método da subclasse que irá sobrescrever, deverá ter o modificador de acesso **override**
- O exemplo a seguir mostra classes responsáveis por operações aritméticas que utilizam sobrescrita.

Polimorfismo

- *Exemplo*

```
namespace ExemploOperacao
{
    class Soma : OperacaoMatematica
    {
        public override double calcular(double x, double y) { return x + y; }
    }
}

namespace ExemploOperacao
{
    class Multiplicacao : OperacaoMatematica
    {
        public override double calcular(double x, double y) { return x*y; }
    }
}

namespace ExemploOperacao
{
    class OperacaoMatematica
    {
        public virtual double calcular(double x, double y) { return 0; }
    }
}
```

```
namespace ExemploOperacao
{
    class OperacaoTest
    {
        public static void calcule(OperacaoMatematica o , double x, double y)
        {
            Console.WriteLine(o.calcular(x,y));
        }

        static void Main(string[] args)
        {
            calcule(new Soma(), 5, 5);
            calcule(new Multiplicacao(), 5, 5);
            Console.ReadKey();
        }
    }
}
```

```
10
25
```

Serialização

- Muitas aplicações necessitam de armazenar ou transferir objetos.
- Para fazer as tarefas simples o possível, o .NET Framework inclui muitas técnicas de serialização.
- As técnicas para converter objetos em binário, ou documentos XML (Extensible Markup Language) que podem ser facilmente armazenados, transferidos e recuperados.

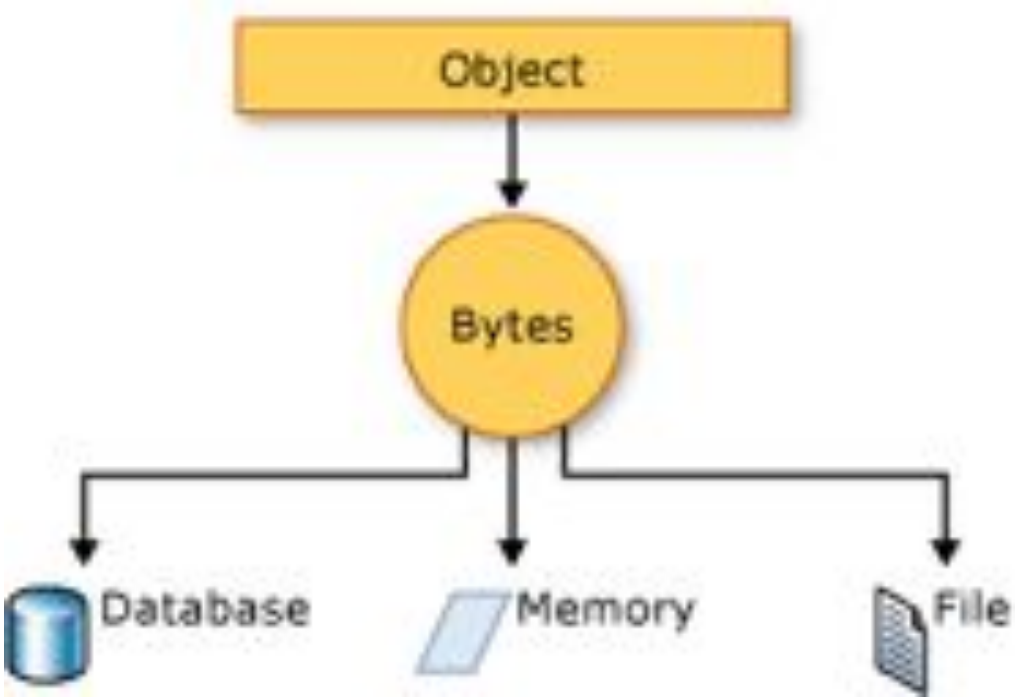
Serialização

- Esta conversão é chamada de **Serialização**.
- Em nosso dia-a-dia de desenvolvimento pode compreender dois termos:

Serialização	Desserialização
• Converter a instância de uma determinada classe para arquivo;	• Converter (recriar) do arquivo (meio) serializado para a instância da classe com os mesmos atributos previamente serializados.

Serialização

- *Visão Geral*



Serialização de objetos

- Como serializar um objeto para um arquivo binário?

```
1   string data = "This must be stored in a file.";
2
3   //Criar arquivo para salvar os dados em
4   FileStream fs = new FileStream("SerializedString.Data", FileMode.Create);
5
6   //Criar um objeto de BinaryFormatter para executar a serialização
7   BinaryFormatter bf = new BinaryFormatter();
8
9   //Use o objeto BinaryFormatter para serializar os dados para o arquivo
10  bf.Serialize(fs, data);
11
12  //Feche o arquivo
13  fs.Close();
14
```

Serialização de objetos

- Como desserializar um objeto para um arquivo binário?

```
1   //Abre o arquivo do qual deseja ler os dados
2   FileStream fs = new FileStream("SerializedString.Data", FileMode.Open);
3
4   //Criar um objeto de BinaryFormatter para realizar a desserialização
5   BinaryFormatter bf = new BinaryFormatter();
6
7   //Cria o objeto para armazenar dados serializados
8   string data = "";
9
10  //Use o objeto BinaryFormatter para desserializar os dados do arquivo
11  data = (string)bf.Deserialize(fs);
12
13  //Feche o arquivo
14  fs.Close();
15
16  // Mostrar a string desserializadas
17  Console.WriteLine(data);
18
```

Serialização de XML

- O .NET Framework inclui uma série de bibliotecas para leitura e gravação de arquivos XML, incluindo os namespaces:
  - System.Xml;
  - System.Xml.Serialization.
- A serialização provê métodos para converter objetos, incluindo os baseados em classes customizadas, para e de arquivos XML.

Serialização de XML

- Com a serialização XML, você pode escrever qualquer objeto para um arquivo texto para depois obter com somente algumas linhas de código.

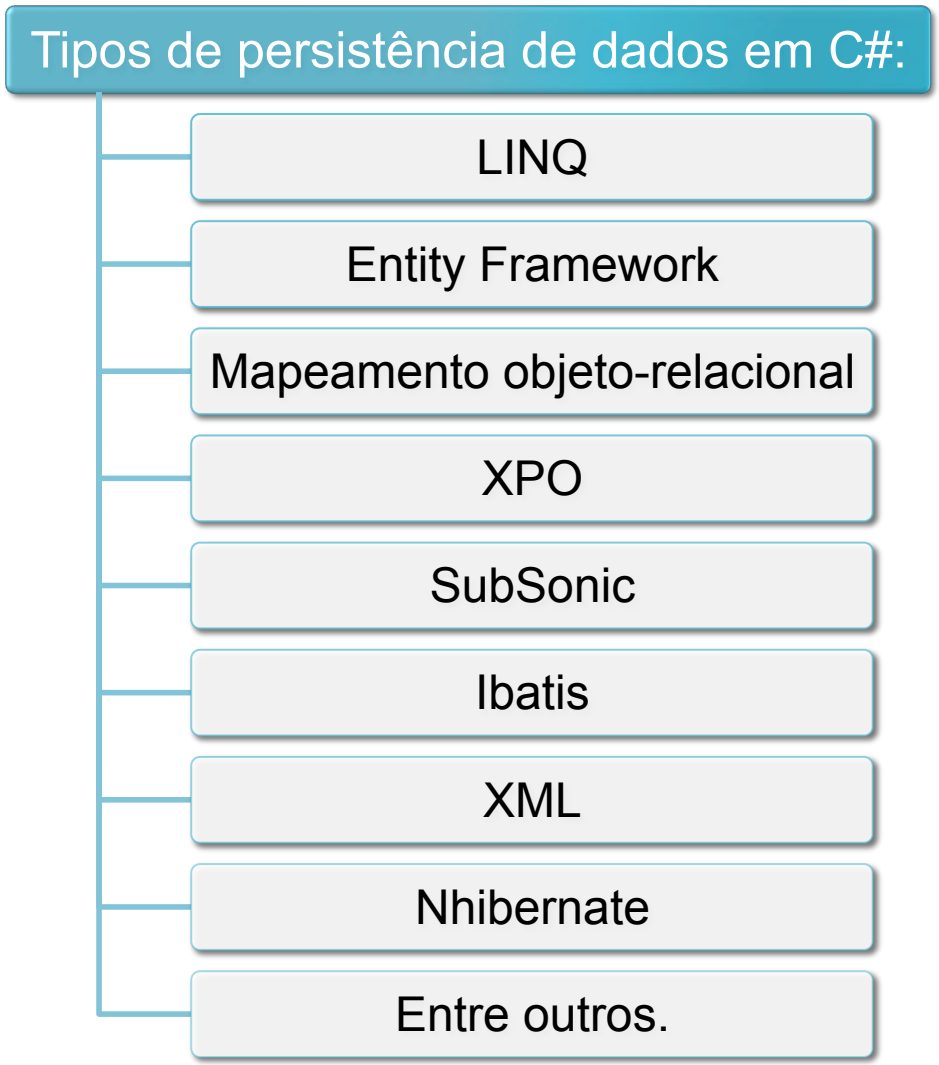
- Similarmente, você pode usar serialização XML para transmitir objetos entre computadores através de WebServices mesmos se os ambos os computadores não estiver usando o .NET Framework.

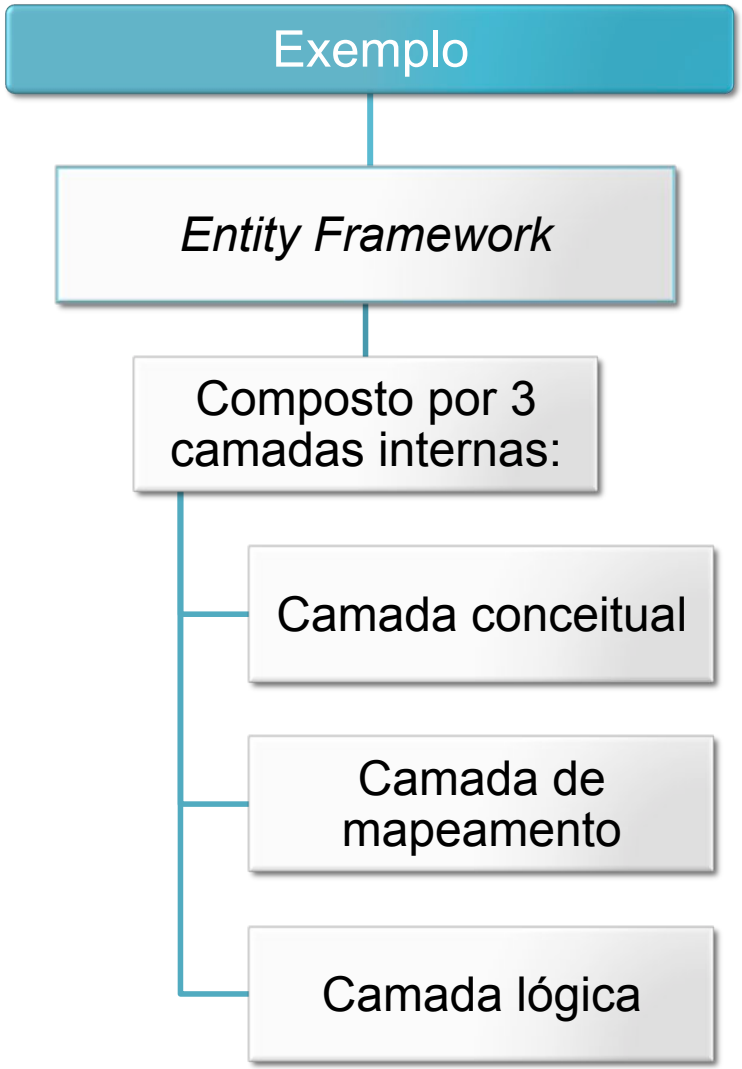
Persistência de Dados

- Quando falamos em **persistência de dados** pensamos no armazenamento "eterno" dos dados, ou seja, enquanto o dispositivo físico de armazenamento dure.

- Persistência de dados envolve um meio físico que permita recuperação de dados, como um banco de dados, um arquivo em disco, etc.

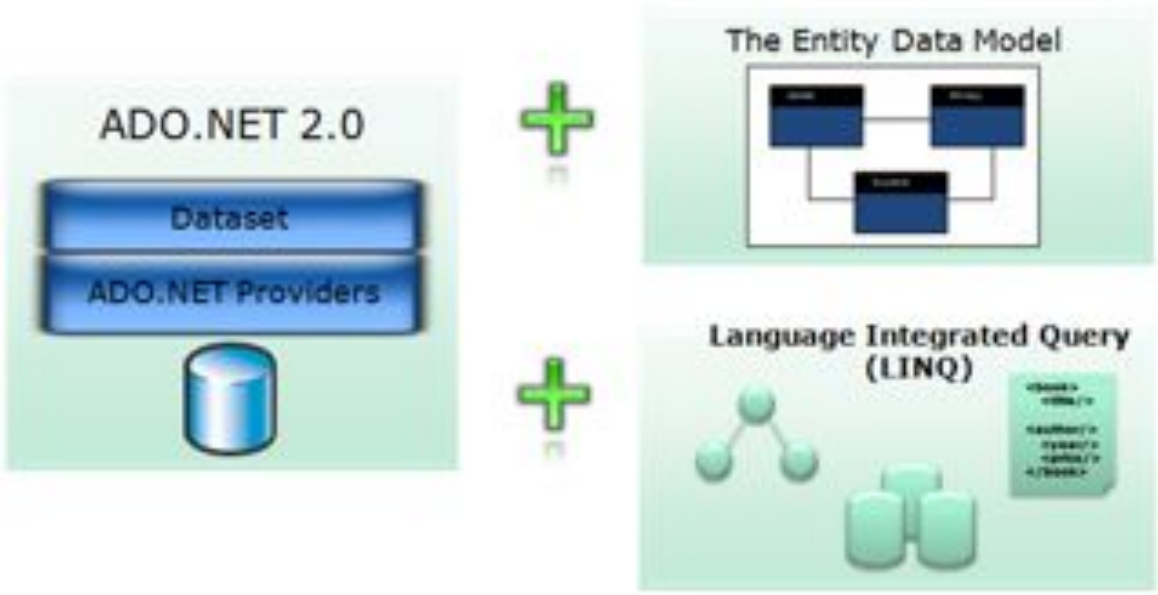
Persistência de Dados





Persistência de Dados

- O **ADO.NET Entity Framework** é um pacote que integra as funcionalidades do ADO.NET 2.0 (com seus mecanismos de *DataSet, DataAdapter, DbConnection, DbCommand* , etc), adicionando 2 novos componentes: o *Entity Data Model* e o *LINQ - Language Integrated Query*.



Threads

- Threads são a base da aplicação de alta performance;
- No .NET Framework, o namespace ***System.Threading*** contém os tipos que são usados para criar e gerenciar múltiplas thread na aplicação.
- Para criar threads você precisa conhecer e utilizar a classe ***Thread***.

Criando Threads

- Como criar uma Thread simples e executá-la?

```
1   ...
2   public static void SimpleWork(){
3       Console.WriteLine("Thread: {0}",
4       Thread.CurrentThread.ManagedThreadId);
5   }
6   ...
7   ThreadStart operacao = new ThreadStart(SimpleWork);
8   Thread theThread = new Thread(operacao);
9   theThread.Start();
10
```

Criando Threads

- Para abortar a execução de uma thread como no exemplo acima, é necessário utilizar o comando theThread.Abort();
- A Classe Threading do .NET possui em média 47 métodos para manipulação de threads, sejam eles Sleep, Suspend, Priority, ThreadState, Name, CurrentContext, entre outros.
- Dentro da Classe Threading existe também semaforos que podem ser utilizados pelo seguinte namespace System.Threading.Semaphore.

Thread com semáforo em C#

Alguns métodos importantes:

Semaphore(Int32, Int32)	Inicializa uma nova instância da Semaphore classe, especificando o número máximo de entradas simultâneas e, opcionalmente, reservando algumas entradas.
WaitOne()	Bloquear o segmento atual até que WaitHandle atual receber um sinal. (Herdado de WaitHandle.)
Release()	Sai do semáforo e retorna a contagem anterior.

Exemplo:

```
1.  using System;
2.  using System.Threading;
3.
4.  namespace threadingSemaphore
5.  {
6.      class Exemplo_Semaforo
7.      {
8.          static Thread[] threads = new Thread[5];
9.          static Semaphore _pool = new Semaphore(2, 2);
10.         static void Worker()
11.         {
12.             Console.WriteLine("{0} está esperando na linha...", Thread.CurrentThread.Name);
13.             _pool.WaitOne();
14.             Console.WriteLine("{0} entrou na região!", Thread.CurrentThread.Name);
15.             Thread.Sleep(300);
16.             Console.WriteLine("{0} saiu da região", Thread.CurrentThread.Name);
17.             _pool.Release();
18.         }
19.         static void Main(string[] args)
20.         {
21.             for (int i = 0; i < 5; i++)
22.             {
23.                 threads[i] = new Thread(Worker);
24.                 threads[i].Name = "thread_" + i;
25.                 threads[i].Start();
26.             }
27.             Console.Read();
28.          }
29.     }
30. }
```

Saída:

```
thread_0 está esperando na linha...
thread_0 entrou na região!
thread_1 está esperando na linha...
thread_1 entrou na região!
thread_2 está esperando na linha...
thread_3 está esperando na linha...
thread_4 está esperando na linha...
thread_0 saiu da região
thread_1 saiu da região
thread_3 entrou na região!
thread_2 entrou na região!
thread_3 saiu da região
thread_4 entrou na região!
thread_2 saiu da região
thread_4 saiu da região
```

- Não há uma ordem garantida, como FIFO ou LIFO, na qual as threads bloqueadas insere o semáforo.

Exceções

- Hierarquia de exceções:
  - Exception é a classe base para exceções
  - Várias classes derivam diretamente de Exception incluindo ApplicationException e SystemException
  - Essas duas classes constituem a base para quase todas as exceções de tempo de execução

Exceções

- Diversos erros já estão definidos na plataforma .NET
- As classes que modelam os erros pré-definidos, derivam da classe SystemException
- Classes derivadas de SystemException:

Exception	Descrição
DivideByZeroException	Erro gerado quando dividimos números inteiros por zero.
IndexOutOfRangeException	Erro gerado quando acessamos posições inexistentes de um array
NullReferenceException	Erro gerado quando utilizamos referências nulas
InvalidCastException	Erro gerado quando realizamos um casting incompatível

Lançando erros

- Para lançar um erro deveremos criar um objeto que deriva da classe Exception

- Após criar uma exception podemos lançar a referencia dela utilizando o comando **throw**

```
if(valor < 0)
{
  System.ArgumentException erro = new System.ArgumentException();
  throw erro;
}
```

Capturando erros

- Para capturar o utilizamos o comando **try-catch**.

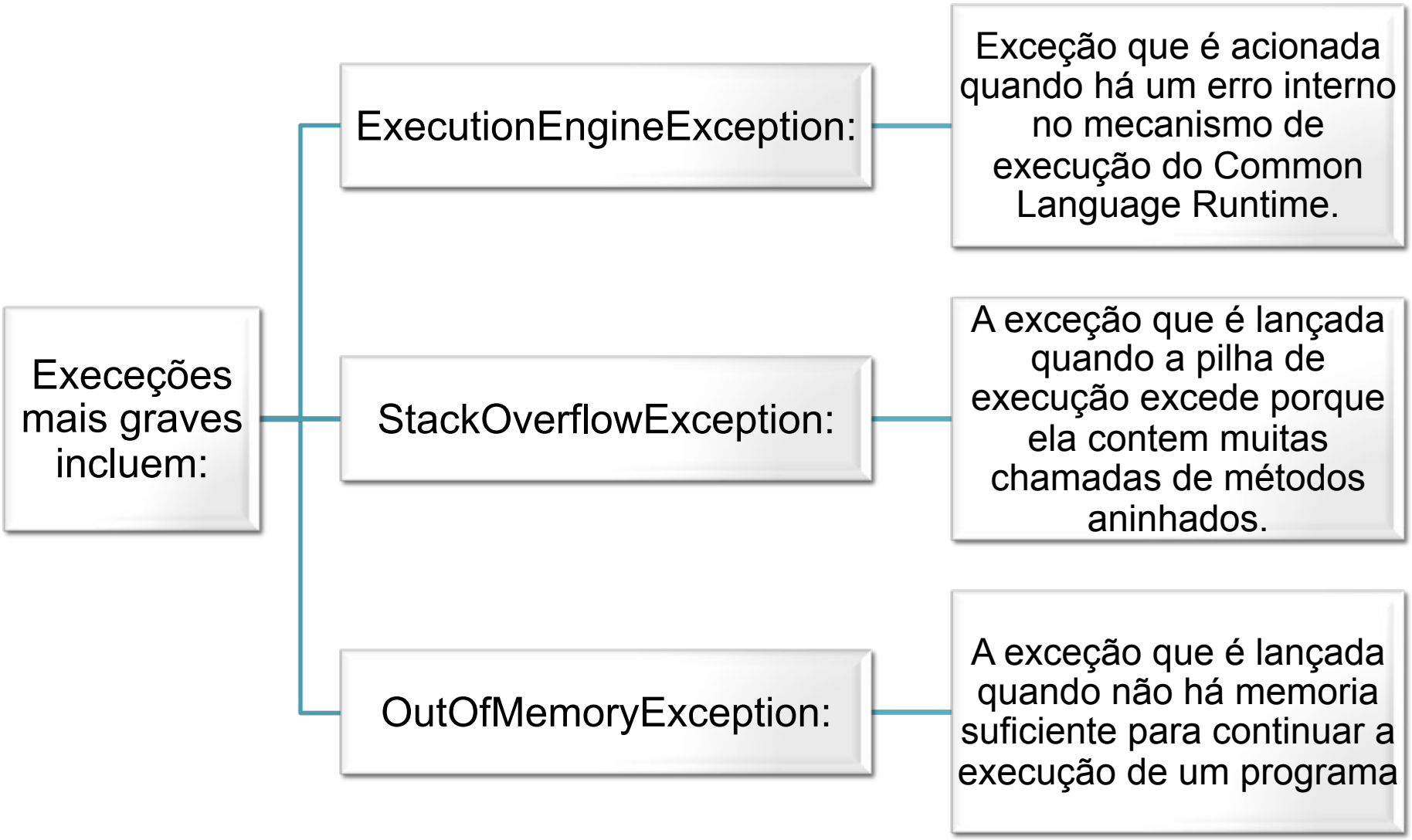
```
class Teste
{
  static void Main()
  {
    Conta c = new Conta();

    try
    {
      c.Deposita(100);
    }
    catch (System.ArgumentException e)
    {
      System.Console.WriteLine("Houve uma System.ArgumentException ao depositar");
    }
    catch (System.IO.FileNotFoundException e)
    {
      System.Console.WriteLine("Houve um FileNotFoundException ao depositar");
    }
  }
}
```

Finally

- Como é comum em Lps, caso queremos executar códigos independentemente se houver erros ou não, usamos o **finally**.

```
try
{
  // código
}
catch(System.DivideByZeroException e)
{
  System.Console.WriteLine("Tratamento de divisão por zero");
}
catch(System.NullReferenceException e)
{
  System.Console.WriteLine("Tratamento de referência nula");
}
finally
{
  // código que deve ser sempre executado
}
```

Exceções


Execeções mais graves incluem:
ExecutionEngineException:
Exceção que é acionada quando há um erro interno no mecanismo de execução do Common Language Runtime.
StackOverflowException:
A exceção que é lançada quando a pilha de execução excede porque ela contem muitas chamadas de métodos aninhados.
OutOfMemoryException:
A exceção que é lançada quando não há memoria suficiente para continuar a execução de um programa

Hierarquia de exceções

- A tabela a seguir ilustra algumas exceções de tempo de execução.

Tipo de exceção	Tipo base	Descrição	Exemplo
Exceção	**Objeto**	Classe base para todas as exceções.	Nenhum (use uma classe derivada desta exceção).
SystemException	**Exceção**	Classe base para todos os erros gerados pelo tempo de execução.	Nenhum (use uma classe derivada desta exceção).
IndexOutOfRangeException	**SystemException**	Lançada pelo tempo de execução somente quando uma matriz é indexada incorretamente.	Indexar uma matriz fora do seu intervalo válido: `arr[arr.Length+1]`
NullReferenceException	**SystemException**	Lançada pelo tempo de execução somente quando um objeto nulo é referenciado.	`object o = null;` `o.ToString();`
AccessViolationException	SystemException	Lançada pelo tempo de execução somente quando é acessada memória inválida.	Ocorre ao interoperar com código não gerenciado ou código gerenciado não seguro, e um ponteiro inválido é usado.

Hierarquia de exceções

Tipo de exceção	Tipo base	Descrição	Exemplo
InvalidOperationException	SystemException	Lançada por métodos quando em um estado inválido.	Chamar `Enumerator.GetNext()` depois de remover um `Item` da coleção subjacente.
ArgumentException	SystemException	Classe base para todas as exceções de argumentos.	Nenhum (use uma classe derivada desta exceção).
ArgumentNullException	ArgumentException	Lançada por métodos que não permitem que um argumento seja nulo.	`String s = null;` `"Calculate".IndexOf (s);`
ArgumentOutOfRangeException	ArgumentException	Lançada por métodos que verificam se os argumentos estão em um determinado intervalo.	`String s = "string";` `s.Chars[9];`
ExternalException	SystemException	Classe base para exceções que ocorrem ou são direcionados para ambientes fora do tempo de execução.	Nenhum (use uma classe derivada desta exceção).
ComException	ExternalException	Exceção que encapsula informações COM HRESULT.	Usada em interoperabilidade COM.

Avaliação da LP

Confiabilidade:

- A desalocação de memória é feita via Garbage Collector
- Linguagem fortemente tipada

Flexibilidade:

- Permite o uso de ponteiros

Redigibilidade:

- Quando implementamos uma interface ou estendemos de uma classe usamos o símbolo “**:**” isso aumenta a redigibilidade em detrimento a legibilidade