Introdução

Curso C#

O que é .NET?

Introdução

O que é .NET?

- Plataforma de desenvolvimento gratuito de código aberto para compilar vários tipos de aplicativos, por exemplo:
  - Aplicativos Web, API ou microserviços
  - Aplicativos Mobile
  - Aplicativos Desktop
    - Windows WPF
    - Windows Forms
  - Jogos
  - Aplicativos Console
  - Serviços do Windows

O que é .NET?

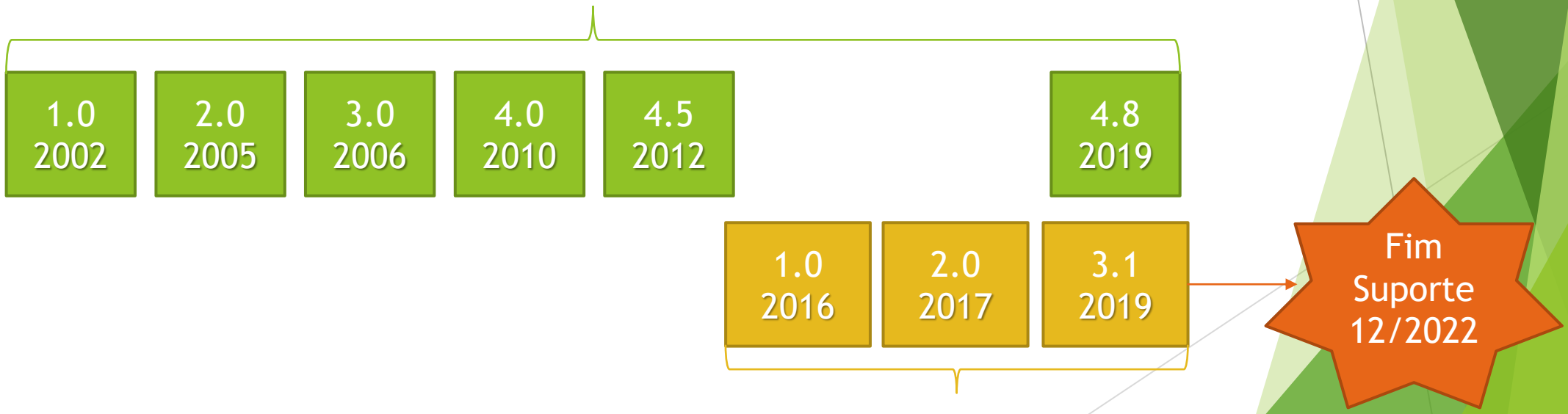
- Multiplataforma
  - Disponível em vários sistemas operacionais, como:
    - Windows
    - macOS
    - Linux
    - Android
    - iOS
    - tvOS
    - watchOS

O que é .NET?

- Multiplataforma
  - Compatível com as seguintes arquiteturas:
    - x64
    - x86
    - ARM32
    - ARM64
  - Permite utilizar API's específicas de sistema operacional. Como o Windows Forms ou WPF em Windows ou plataformas mobile utilizando o Xamarin.

O que é .NET?

- História
  - O .NET Framework foi criado em um princípio como uma resposta da Microsoft contra o JAVA que já levava vários anos no mercado.
  - A primeira versão (1.0) foi lançada no ano de 2002.



O que é .NET?

- História
  - O .NET 5 foi lançado em Novembro/2020

Linguagem C#

Introdução

Linguagem C#

- A linguagem C# foi criada juntamente com a arquitetura da plataforma .NET
- Linguagem influenciada principalmente por C++ e Java
- Seu principal engenheiro foi Anders Hejlsberg, também criador do Turbo Pascal e Delphi

Linguagem C#

- Principais características:
  - Sintaxe simples e de fácil aprendizagem
  - Muito familiar a Java e C
  - Multiplataforma
  - Tratamento de erros
  - Bibliotecas
  - Melhor gestão de memória

Plataformas

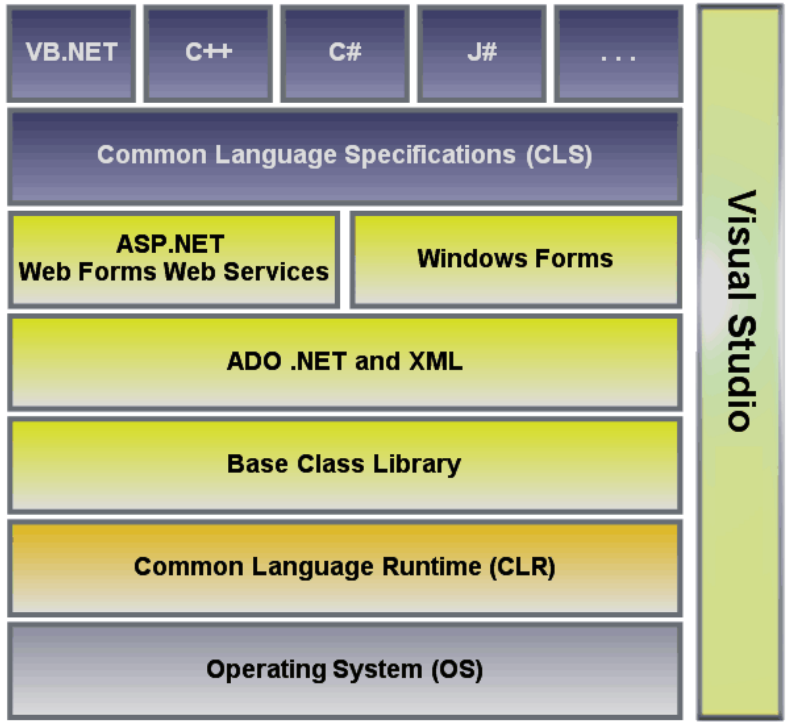
Introdução

Plataformas

- De uma forma simples e compreensível podemos dizer que:

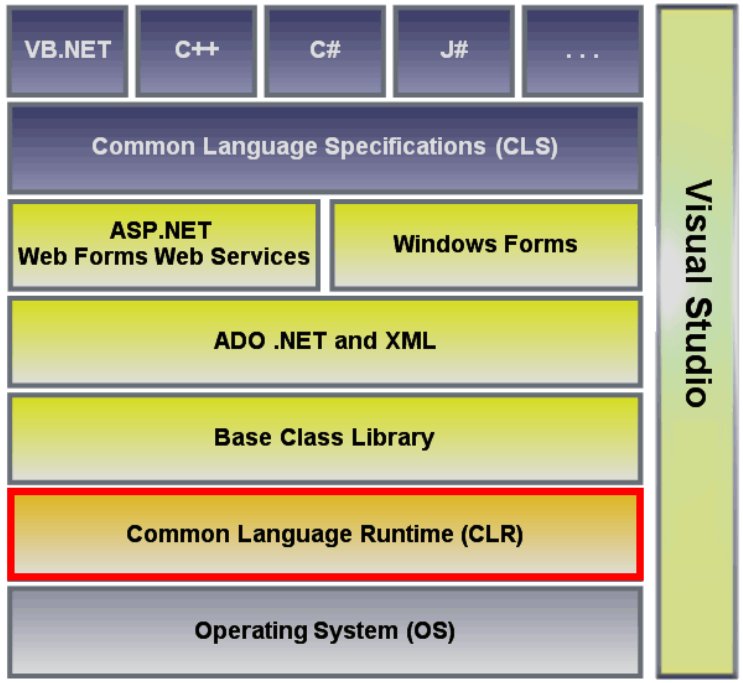
A plataforma .NET é um amplo conjunto de bibliotecas de desenvolvimento, que podem ser utilizadas com o principal objetivo de **acelerar o desenvolvimento de software** e obter de forma automática características avançadas de segurança, rendimento, etc.

Plataformas



Plataformas - CLR

- O CLR ou Common Language Runtime é a parte do .NET responsável de executar os programas desenvolvidos para a plataforma.

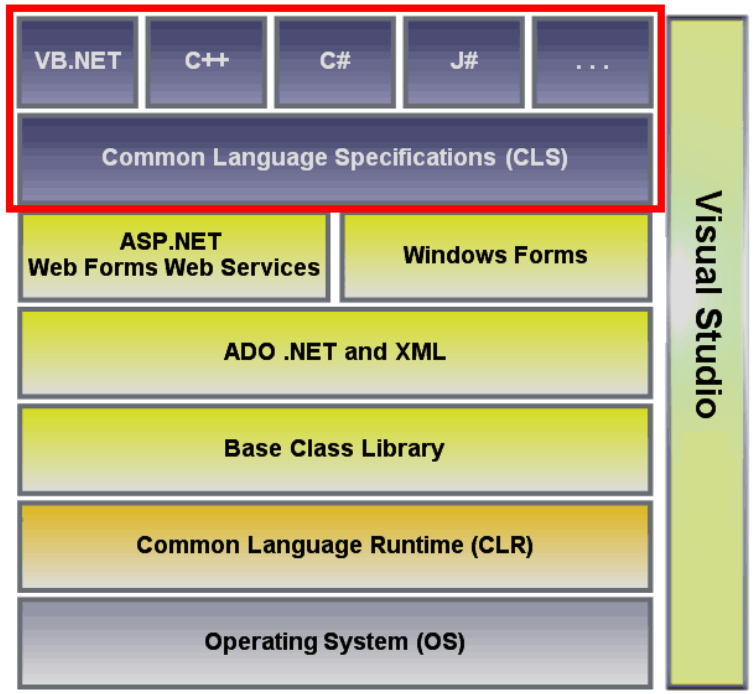


Plataformas - CLR

- O CLR é responsável de compilar uma forma de código IL (Linguagem intermediário) ao código de máquina nativo mediante um processo chamado JITC (Just In Time Compilation)
- Dessa forma garante um código desenvolvido em uma máquina possa ser executado em outra.

Plataformas - CLS

- CLS - Common Language Specifications define um conjunto de características que são necessárias em muitas aplicações comuns, para assim ser compatível entre diferentes linguagens de programação.

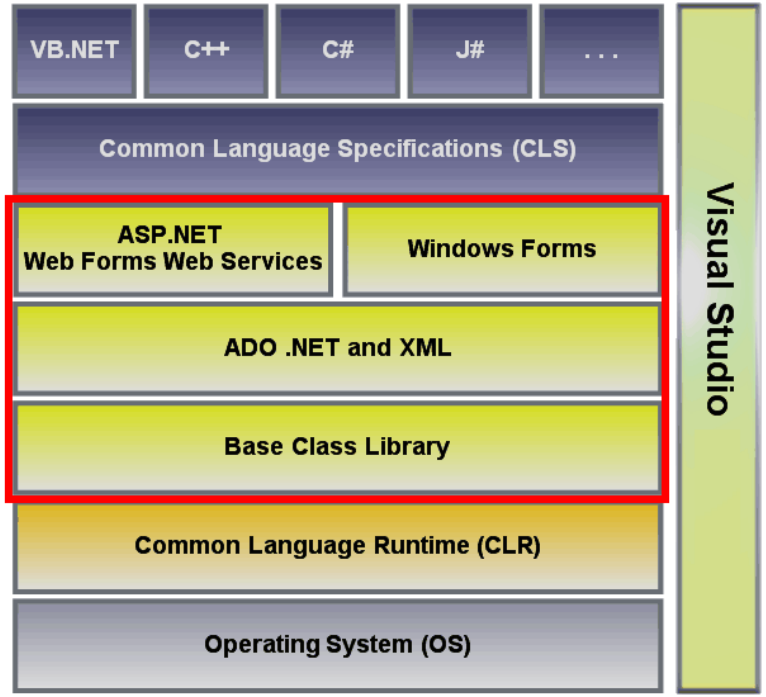


Plataformas - CLS

- Ao contrario de outras plataformas, o .NET não está preso a uma determinada linguagem, mas as principais são C# e Visual Basic.
- Qualquer componente criado com alguma linguagem, pode ser utilizado de forma transparente desde qualquer outra linguagem em .NET
- CLS está formada por um conjunto de regras que devem ser seguidas pelas definições de tipos de dados, para que assim possam interagir entre diferentes tipos de linguagem.

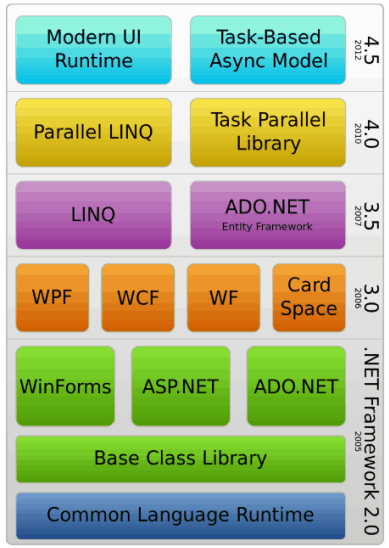
Plataformas - BCL

- BCL - Base Class Library, está formada por bibliotecas ou APIs especializadas que podem ser utilizadas por qualquer tipo de linguagem de programação em .NET



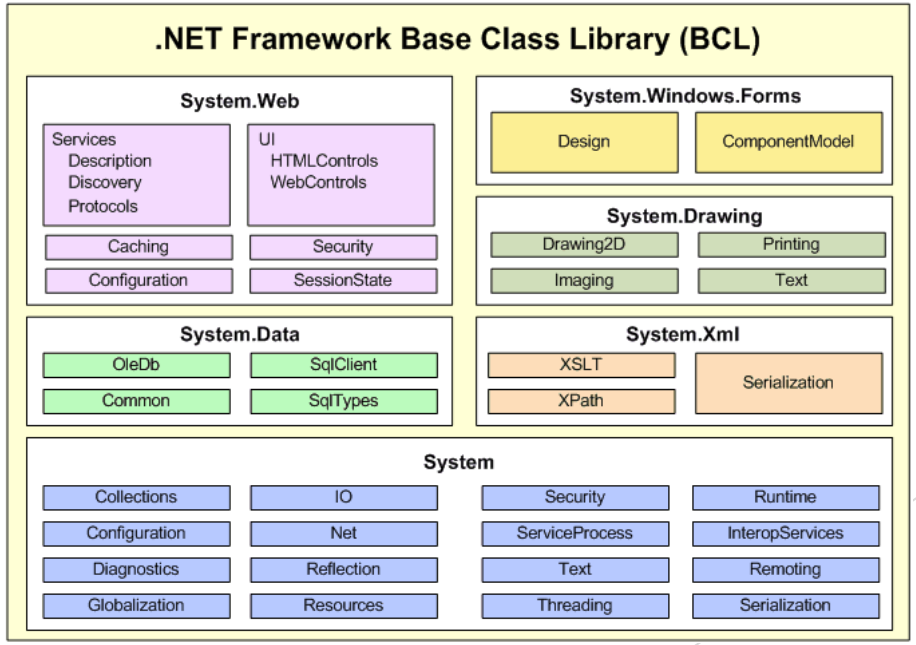
Plataformas - BCL

- Cada uma dessas bibliotecas podem conter incontáveis classes que contem vários métodos e funções com características concretas.
- Dessa forma, podemos encontrar bibliotecas com funcionalidades para quase qualquer coisa: enviar e-mail, escrever arquivos de texto, acessar bancos de dados, criptografia, etc...
- As BCLs foram evoluindo junto com o .NET, a cada nova versão nossas bibliotecas foram criadas ou melhoradas.



Plataformas - BCL

- Dentro da BCL, é possível encontrar inúmeras classes diferentes, agrupadas de forma organizada pelos Namespaces.
- Un Namespace nada mais é que um identificador que permite organizar, todas as suas classes e separar-las dentro do código.
- Um pequeno exemplo seriam:



TryParse

Curso C#

TryParse

- Parse vs TryParse
- Double.Parse(string) ou int.Parse(string) Datetime.parse(string)
  - Parse: lança uma exceção se a conversão falhar.
  - TryParse: Se a conversão falhar retorna zero.

- Sintaxe:

```
//Exemplo
String a algo
int.TryParse("A", out int mes);

Resultado: Falso e mes = 0
```

```
//Exemplo

int.TryParse("121", out int valor);

Resultado: Verdadeiro e valor = 121
```

Operador Ternário

Curso C#

Operador Ternário

- Utilizado para validar uma condição.
- Sintaxe: *condição ? Expressão True : Expressão False*
  - Resultado será um booleano

```
//usando if...else

If (hora > 12){
 saudacao = “Boa tarde”;
} else {
 saudação = “Bom dia”;
}
```

```
//usando operador ternário


Saudacao = hora > 12 ? “Boa tarde” : “Bom dia”;
```

Exceções

Curso C#

Exceções

- Exceções são erros durante a execução de um programa que saem do controle do programador.

- Erros de leitura ou escritura
- Problemas de memoria
- Acesso a arquivos inexistentes
- Conexões com o Banco de Dados
- Estouro de Memória
- ETC

Exceções

- Exceções são erros durante a execução de um programa que saem do controle do programador.

Bloco try … catch

Try – Tentavia
Catch - Captura

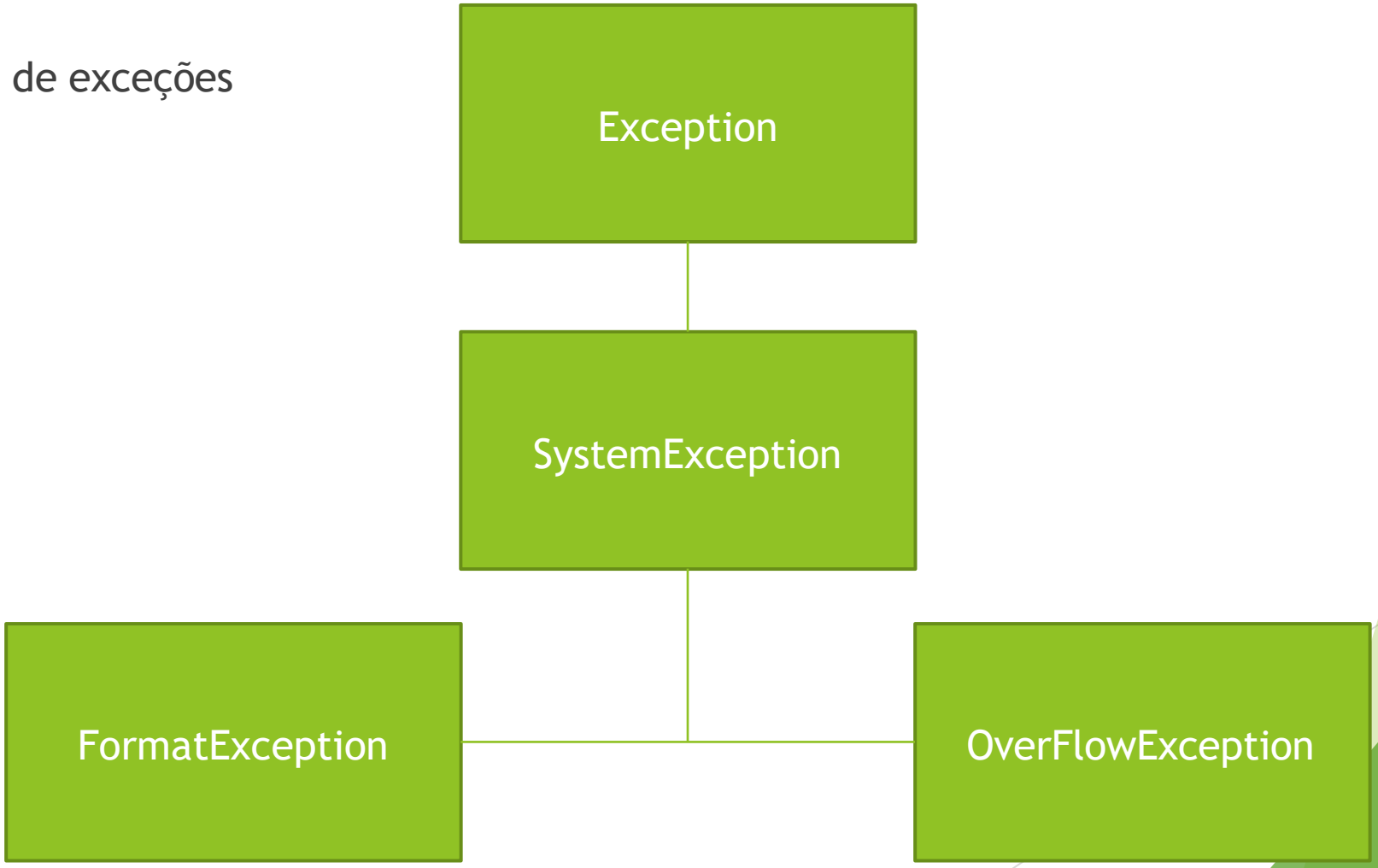
Exceções

- É possível capturar várias exceções e vários tipos de exceções.

```
Try
{
   //código do sistema
}catch(FormatException ex){
   //código que se executa
     depois da exceção
}catch(OverFlowException ex){
     //código depois do
     OverFlowException
}
```

Exceções

- Herança de exceções



Exceções

- Conflitos de uso em vários catch

```
23                  try
24                  {
25                      meuNumero = int.Parse(Console.ReadLine());
26                  }catch(Exception ex)
27                  {
28                      meuNumero = -1;
29                      Console.WriteLine(" - Exception" );
30                      Console.WriteLine(ex.Message);
31                  }catch(FormatException ex)
32                  {
33                      meuNumero = -1;
34                      Console.WriteLine(" - Format Exception" );
35                      Console.WriteLine(ex.Message);
36
```

```
PROBLEMS 1   OUTPUT   TERMINAL   DEBUG CONSOLE

O número é maior.
PS C:\Users\rodri\Desktop\Projetos\JogoAleatorio> dotnet run
C:\Users\rodri\Desktop\Projetos\JogoAleatorio\Program.cs(31,24): error CS0160: Cláusula catch anterior já captura todas as exceções desta ou
 de um super tipo ("Exception") [C:\Users\rodri\Desktop\Projetos\JogoAleatorio\JogoAleatorio.csproj]
C:\Users\rodri\Desktop\Projetos\JogoAleatorio\Program.cs(36,24): error CS0160: Cláusula catch anterior já captura todas as exceções desta ou
 de um super tipo ("Exception") [C:\Users\rodri\Desktop\Projetos\JogoAleatorio\JogoAleatorio.csproj]
```

Exceções

- Expressão checked
  - Serve para controlar um estouro aritmético.

Em um contexto não verificado, o estouro aritmético é ignorado, e o resultado é truncado descartando todos os bits de ordem superior que não se encaixam no tipo de destino.

  - Sintaxe:

```
Checked {
        //Código a ser verificado.
}
```

Exceções

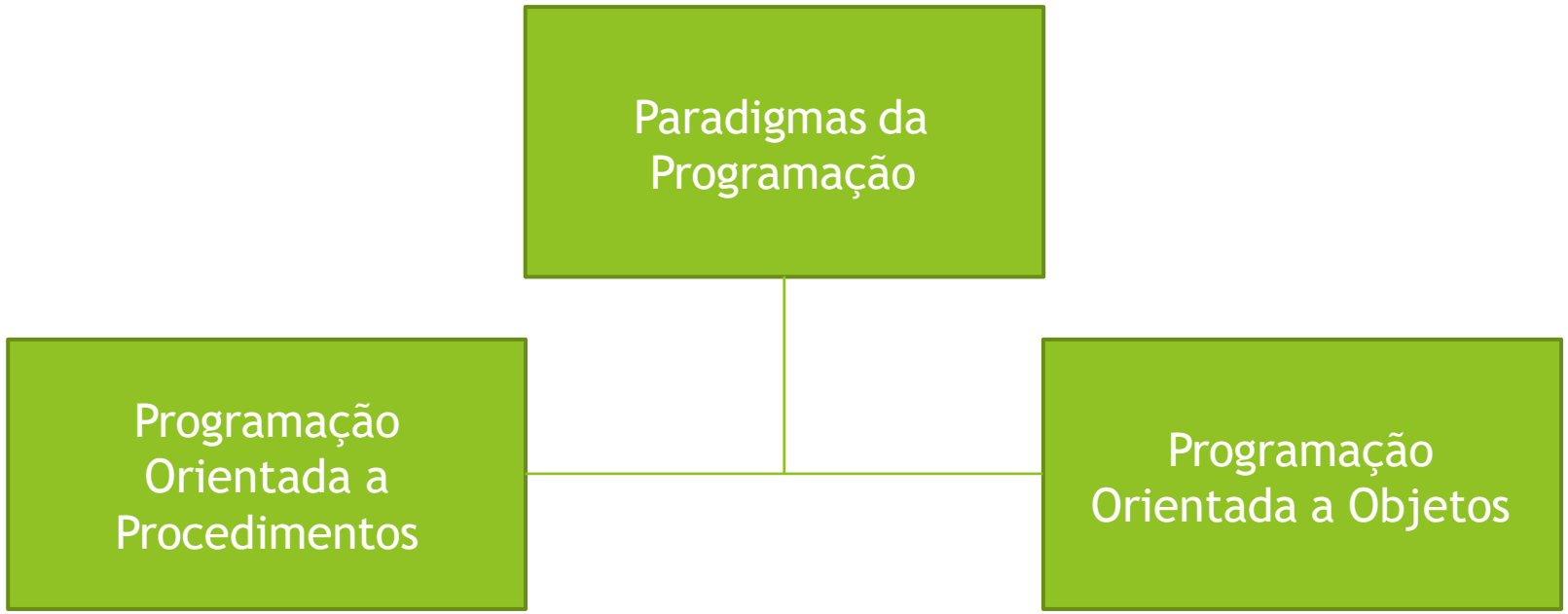
- Conflitos de uso em vários catch
- Expressão checked
- Execução método Throw
- Uso do Finally

Programação Orientada a Objetos

Curso C#

Programação Orientada a Objetos

- 2 Formas de Programar (Paradigmas)



Programação Orientada a Objetos

- Programação Orientada a Procedimentos
  - Alguns exemplos de linguagens: Fortran, Cobol, Basic, etc.
  - Desvantagens:
    - Códigos muito grandes em programas muito complexos.
    - Códigos complexos são difíceis de decifrar
    - Pouco reutilizável
    - Qualquer erro é bem provável que o programa pare de funcionar
    - Dificuldade de depurar caso seja necessário ou tenha algum erro
    - Uso frequente do “Código Espaguete”

Programação Orientada a Objetos

- Programação Orientada a Procedimentos
  - Alguns exemplos de linguagens: Fortran, Cobol, Basic, etc.
  - Desvantagens:
    - Códigos muito grandes em programas muito complexos.
    - Códigos complexos são difíceis de decifrar
    - Pouco reutilizável
    - Qualquer erro é bem provável que o programa pare de funcionar
    - Dificuldade de depurar caso seja necessário ou tenha algum erro
    - Uso frequente do “Código Espaguete”

```
10 INPUT A$
20 GOTO 200
30 PRINT A$,B
40 GOTO 1000
100 GOTO 30
200 INPUT B
210 IF B>=0 GOTO 30
220 IF B<0 GOTO 100
500 GOTO 3000
1000 INPUT C$
1200 INPUT D
2000 IF D>0 GOTO 500
3000 PRINT A$,"+",C$,"=",B+D
5000 END
```

Programação Orientada a Objetos

- Programação Orientada a Procedimentos
  - Alguns exemplos de linguagens: Fortran, Cobol, Basic, etc.
  - Desvantagens:
    - Códigos muito grandes em programas muito complexos.
    - Códigos complexos são difíceis de decifrar
    - Pouco reutilizável
    - Qualquer erro é bem provável que o programa pare de funcionar
    - Dificuldade de depurar caso seja necessário ou tenha algum erro
    - Uso frequente do “Código Espaguete”

```
10 INPUT A$
20 GOTO 200
30 PRINT A$,B
40 GOTO 1000
100 GOTO 30
200 INPUT B
210 IF B>=0 GOTO 30
220 IF B<0 GOTO 100
500 GOTO 3000
1000 INPUT C$
1200 INPUT D
2000 IF D>0 GOTO 500
3000 PRINT A$,"+",C$,"=",B+D
5000 END
```

```
10 INPUT A$,B
20 PRINT A$,B
30 INPUT C$,D
40 PRINT A$,"+",C$,"=",B+D
50 END
```

Programação Orientada a Objetos

- Programação Orientada a Objetos
  - Tem como objetivo tentar trazer a “vida real” para dentro do nosso programa.

Programação Orientada a Objetos

- Programação Orientada a Objetos
  - Tem como objetivo tentar trazer a “vida real” para dentro do nosso programa.
  - Tendo como base que qualquer objeto da “vida real” tem um estado, um comportamento e umas propriedades.

Programação Orientada a Objetos

- Programação Orientada a Objetos
  - Tem como objetivo tentar trazer a “vida real” para dentro do nosso programa.
  - Tendo como base que qualquer objeto da “vida real” tem um estado, um comportamento e umas propriedades.
  - Exemplo: Um carro
    - Qual é o estado do carro? – Parado, circulando, estacionado, etc.
    - Quais são as propriedades do carro? – Cor, peso, potência, acessórios, etc.
    - Qual o comportamento do carro? – Acelerar, frear, girar, etc.

Programação Orientada a Objetos

- Programação Orientada a Objetos
  - Alguns exemplos de linguagens: C++, Java, .NET, etc.
  - Vantagens:
    - Programas divididos em partes, módulos, classes. Modularização.
    - Códigos reutilizável. Herança
    - Em caso de erro em alguma linha do código, o programa continuará executando. Tratamento das exceções (erros)
    - Encapsulamento

Programação Orientada a Objetos

- Programação Orientada a Procedimentos

Programação Orientada a Objetos

- Programação Orientada a Objetos

Programação Orientada a Objetos

- Programação Orientada a Objetos

Programação Orientada a Objetos

- Programação Orientada a Objetos

Modificadores de acesso:

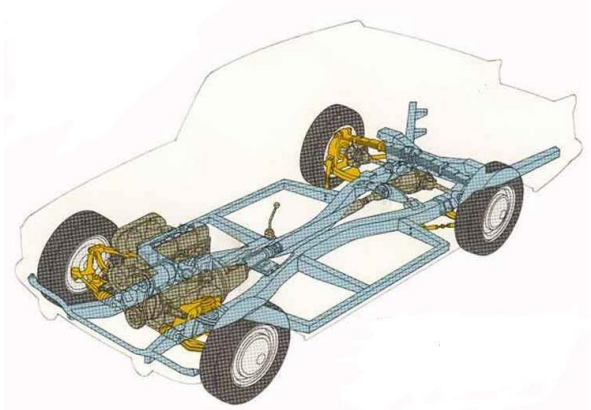
- Public
  - Acessível de qualquer parte
- Private
  - Acessível dentro da própria classe
- Protected
  - Acessível na classe derivada
- Internal
  - Acessível dentro da mesma DLL
- Protected Internal
  - Acessível no assembly atual ou nos tipos que derivam da classe recipiente
- Private Protected
  - Acessível da mesma classe ou classe derivada de outro Assembly
- Default
  - Acessível dentro da mesma classe.

Classes e Objetos

Curso C#

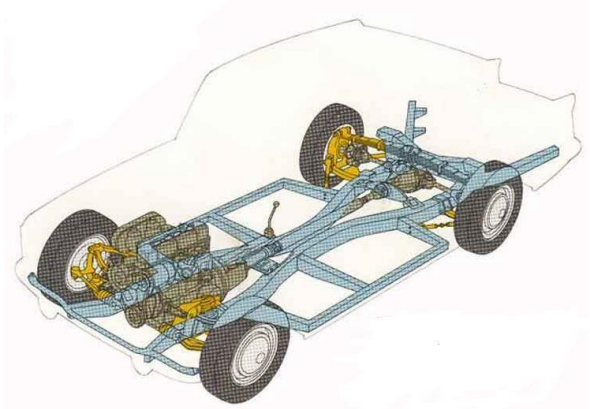
Programação Orientada a Objetos

- Classe
  - Modelo onde são descritas as características comuns de um grupo de objetos.



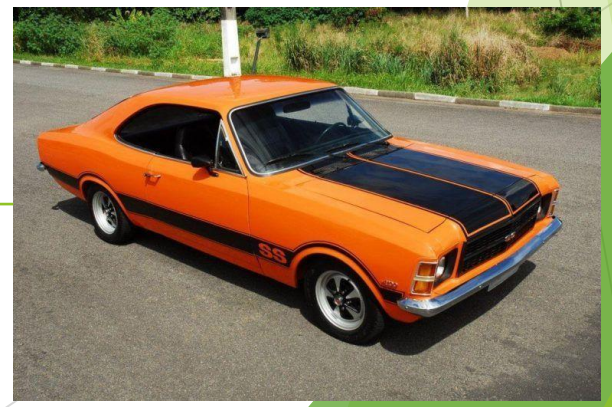
Programação Orientada a Objetos

- Classe
  - Modelo onde são descritas as características comuns de um grupo de objetos.



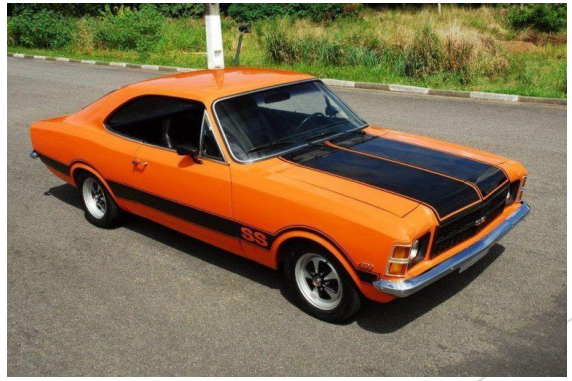
Classe

Objetos



Programação Orientada a Objetos

- Objeto
  - Propriedades (Atributos)
    - Cor
    - Peso
    - Combustível
    - Motor
  - Comportamento (Métodos ou Funções)
    - Acelerar
    - Frear
    - Girar



Programação Orientada a Objetos

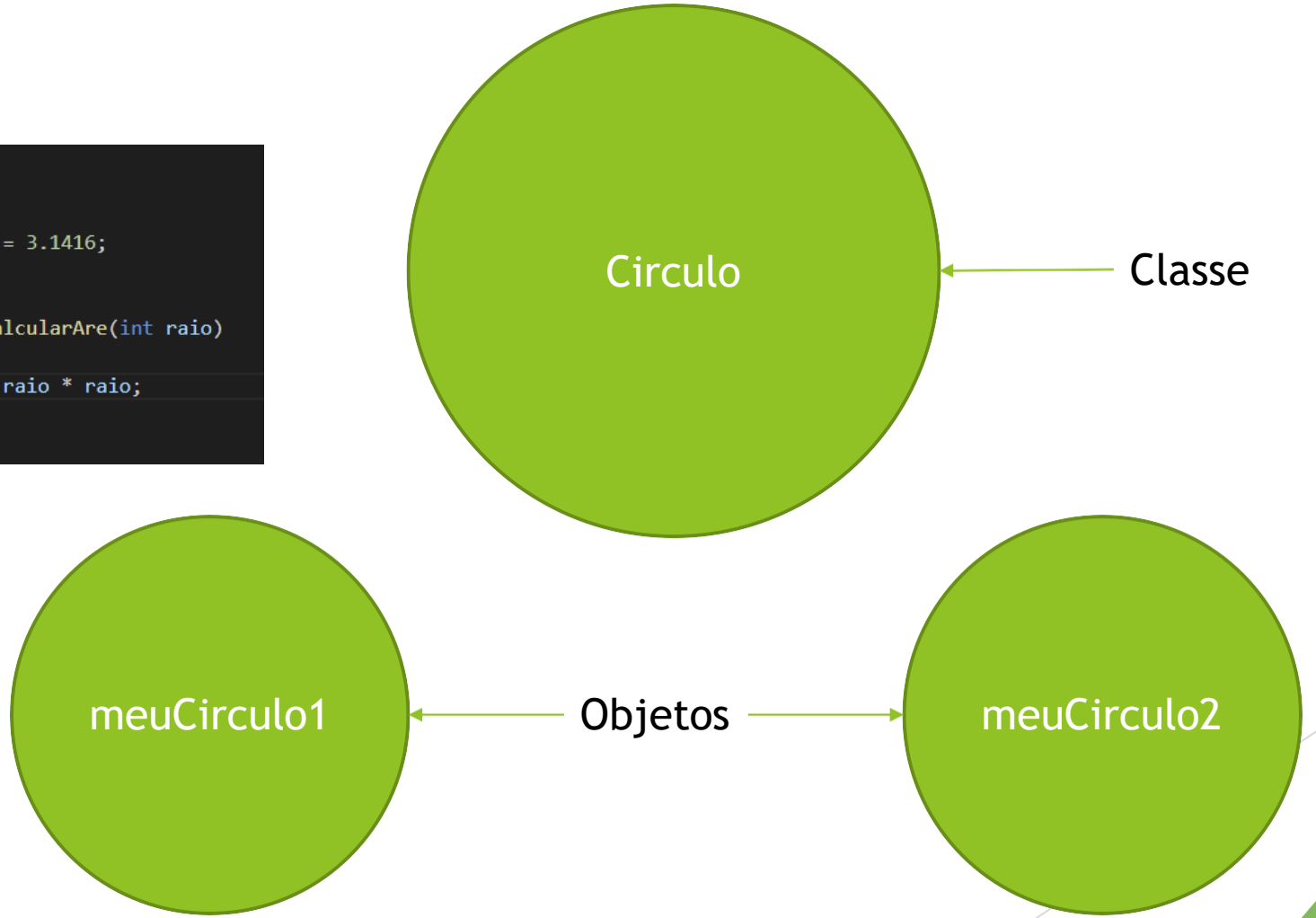
- Objeto
  - Acessar as propriedades do objeto pelo código
    - Maverick.cor = “Amarelo”
    - Maverick.Peso = 1500;
    - Maverick.Combustível = Gasolina;
    - Maverick.Motor = 3.0;
  - Acessar o comportamento (Métodos ou Funções)
    - Maverick.Acelerar();
    - Maverick.Frear();
    - Maverick.Girar();

Programação Orientada a Objetos

- Objeto

```
class Circulo
{
    1 reference
    const double pi = 3.1416;

    0 references
    public double calcularAre(int raio)
    {
        return pi * raio * raio;
    }
}
```



Programação Orientada a Objetos

- Exercício:
  - Criar um programa para fazer o cálculo da área de círculos, quadrado e retângulo.
    - O usuário deverá escolher o que ele deseja calcular
    - Fórmulas:
      - Círculo: Pi x Raio x Raio
      - Quadrado: Lado²
      - Retângulo: Base x Altura

Encapsulamento

Curso C#

Programação Orientada a Objetos

- Encapsulamento
  - Ocultar o estado interno e a funcionalidade de um objeto e permitir o acesso apenas por meio de um conjunto público de funções.

Programação Orientada a Objetos

- Encapsulamento
  - Ocultar o estado interno e a funcionalidade de um objeto e permitir o acesso apenas por meio de um conjunto público de funções.
  - Muitos dos métodos ou variáveis não devem ser visíveis desde outras classes, é uma forma de proteção/segurança.

Programação Orientada a Objetos

- Boas práticas na hora de programar
  - Os identificadores “public” devem começar com letra maiúscula.
    - Norma “PascalCase”
      - Ex.: public double CalcularArea()...
  - Os identificadores que não são “public” devem começar por letra minúscula.
    - Norma “camelCase”
      - Ex. precoProduto

Construtores

Curso C#

Programação Orientada a Objetos

- Construtores
  - Tem como objetivo dar um estado inicial na criação de um objeto, que poderá ser modificado depois.

Programação Orientada a Objetos

- Construtores
  - Tem como objetivo dar um estado inicial na criação de um objeto, que poderá ser modificado depois.
  - É um método com o mesmo nome da Classe, nunca será void e não tem return.

Programação Orientada a Objetos

- Construtores
  - Tem como objetivo dar um estado inicial na criação de um objeto, que poderá ser modificado depois.
  - É um método com o mesmo nome da Classe, nunca será void e não tem return.
  - Em C#, se não está declarado, o compilador cria um vazio por Default.

Getters e Setters

Curso C#

Programação Orientada a Objetos

- Get
  - Método utilizado para retornar um valor do tipo de propriedade. É equivalente à leitura do valor do campo.
- Set
  - Método utilizado para estabelecer propriedades ao objeto. Não devolve nenhum valor, sempre será void.

Métodos e variáveis Static

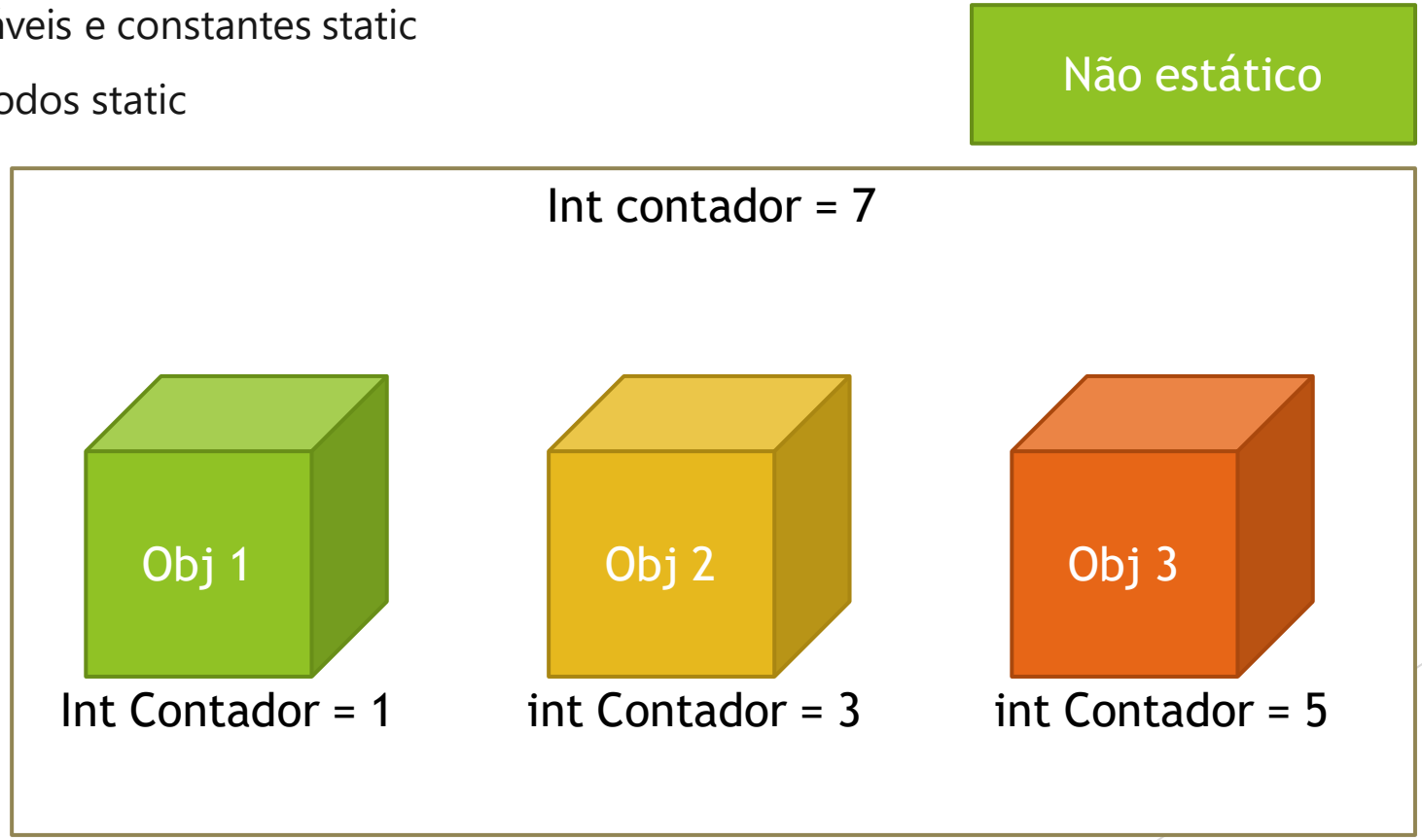
Curso C#

Programação Orientada a Objetos

- Static
  - Variáveis e constantes static
  - Métodos static

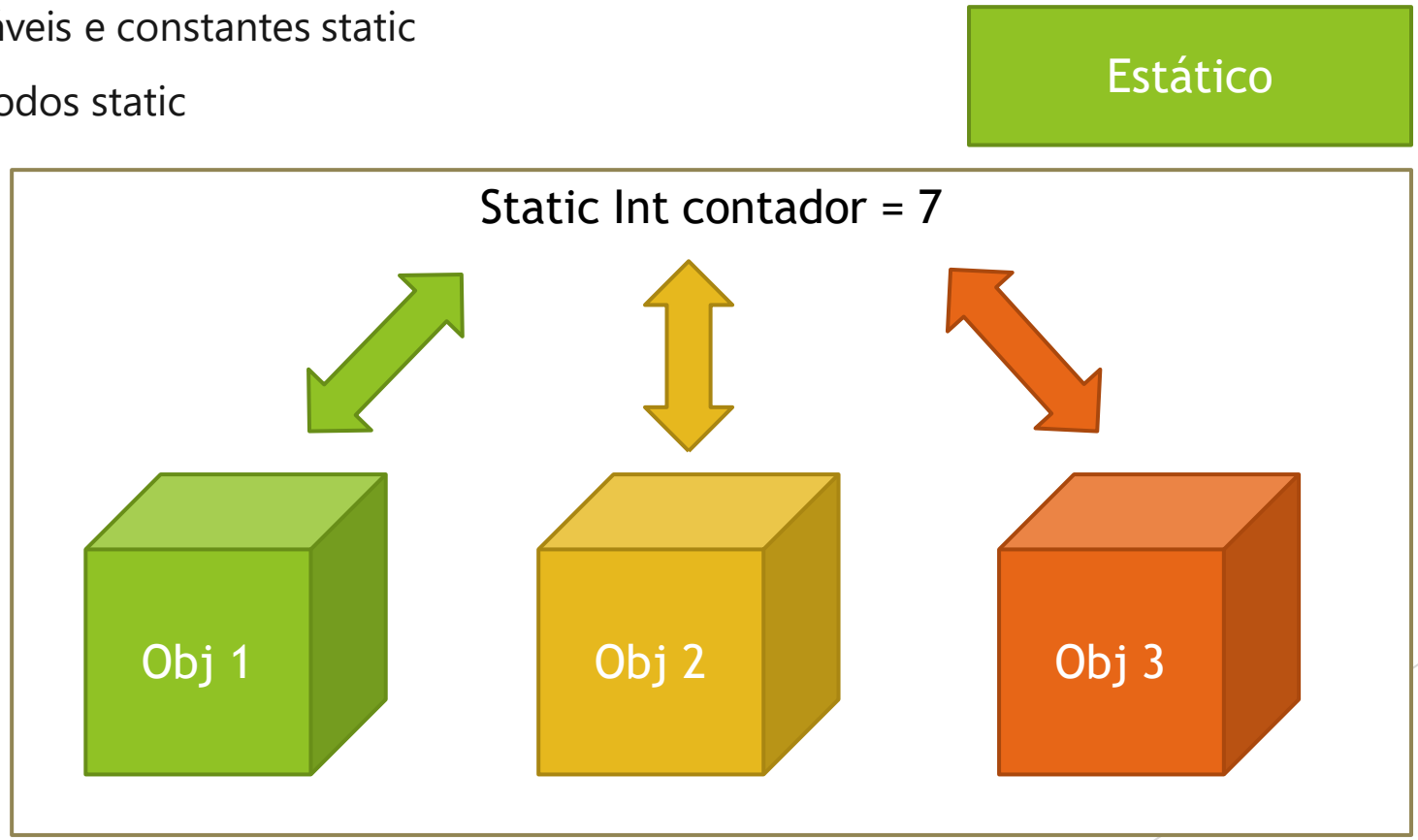
Programação Orientada a Objetos

- Static
  - Variáveis e constantes static
  - Métodos static



Programação Orientada a Objetos

- Static
  - Variáveis e constantes static
  - Métodos static



Classes Anônimas

Curso C#

Programação Orientada a Objetos

- Classes Anônimas
  - Os tipos anônimos fornecem um modo conveniente de encapsular um conjunto de propriedades somente leitura em um único objeto sem a necessidade de primeiro definir explicitamente um tipo.

Programação Orientada a Objetos

- Classes Anônimas
  - Os tipos anônimos fornecem um modo conveniente de encapsular um conjunto de propriedades somente leitura em um único objeto sem a necessidade de primeiro definir explicitamente um tipo.
  - Exemplo:
    - var minhaOutraVariavel = new {Nome="Maria",Idade=25}

Programação Orientada a Objetos

- Classes Anônimas
  - Restrições
    - Possui apenas campos públicos
    - Todos os campos devem ser iniciados
    - Os campos não podem ser static
    - Não é possível definir métodos

Arrays

Curso C#

Arrays

- O que são?
  - Estruturas de dados que contém uma coleção de valores do mesmo tipo.

Arrays

- O que são?
  - Estruturas de dados que contém uma coleção de valores do mesmo tipo.
- Para que são utilizados?
  - Para armazenar valores que normalmente tem alguma relação entre si.

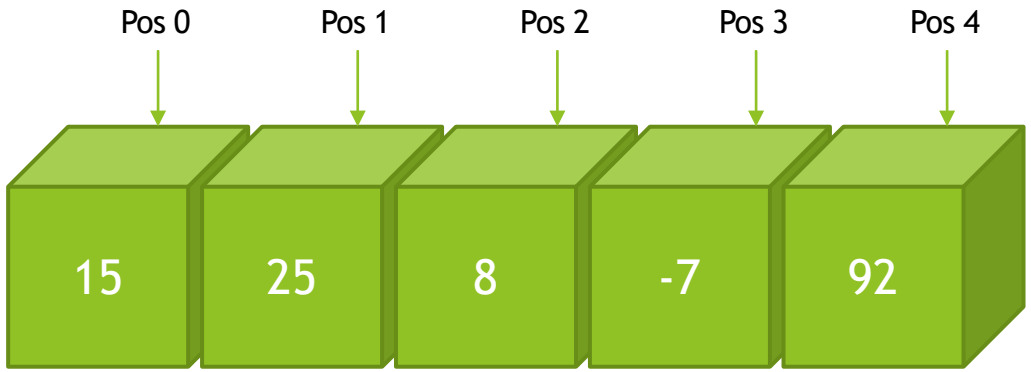
Arrays

- O que são?
  - Estruturas de dados que contém uma coleção de valores do mesmo tipo.
- Para que são utilizados?
  - Para armazenar valores que normalmente tem alguma relação entre si.
- Sintaxe:
  - Declaração: int[] meuArray; string[] meuArrayString;
  - Iniciação: meuArray = new int[4];

    meuArrayString = new string[30];

Arrays

- Armazenar valores Array

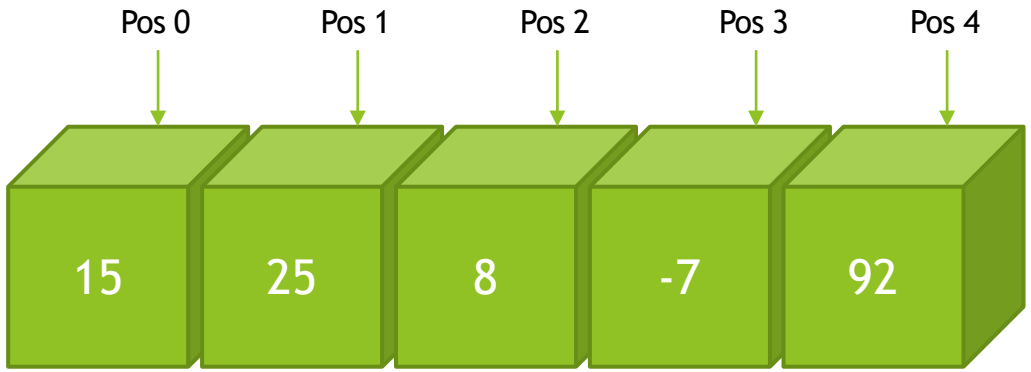



```
meuArray[0] = 15;
meuArray[1] = 25;
meuArray[2] = 8;
meuArray[3] = -7;
meuArray[4] = 92;
```

```
Int[] meuArray = new int[5];
```

Arrays

- Armazenar valores Array




```
meuArray[0] = 15;
meuArray[1] = 25;
meuArray[2] = 8;
meuArray[3] = -7;
meuArray[4] = 92;
```

```
Int[] meuArray = new int[5];
```

Iniciar array com valores: `int[] meuArray = {15,25,8,-7,92}`

Arrays Implícitos

Curso C#

Arrays Implícitos

- Array flexível onde não especificamos o tipo, nem quantos elementos vai ter.
  - Exemplo: var datos = new [] {“João”,”Da Silva”,”Barbosa”};

Arrays Objetos

Curso C#

Arrays

- Como um array primitivo, mas será definido como um objeto.
  - Exemplo: Funcionarios[] arrayFuncionarios = new Funcionarios[2];

Arrays Tipos Anônimos

Curso C#

Arrays

- Array de tipos que não possuem nomes/anônimos
  - Exemplo:

```
var pessoas = new []
{
        new{Nome="João", Idade=19},
        new{Nome="José", Idade=22},
        new{Nome="Maria", Idade=29}
};
```

Arrays e loop ForEach

Curso C#

Arrays e ForEach

- A instrução foreach fornece uma maneira simples e limpa de percorrer através dos elementos de um array.

Arrays e ForEach

- A instrução foreach fornece uma maneira simples e limpa de percorrer através dos elementos de um array.
- Recorrerá todos os elementos do Array, diferente do For onde o programador pode definir o que ele quer filtrar.

Arrays e ForEach

- A instrução foreach fornece uma maneira simples e limpa de percorrer através dos elementos de um array.
- Recorrerá todos os elementos do Array, diferente do For onde o programador pode definir o que ele quer filtrar.

- Exemplo:

```
Foreach(Funcionarios func in arrayFuncionarios)
{
        //Codigo foreach;
}
```