





Gabriel Baltazar

{ Apaixonado por Delphi desde sempre.



gabrielbaltazar.dev@gmail.com



github.com/gabrielbaltazar



linkedin.com/in/gabrielbaltazar-dev/



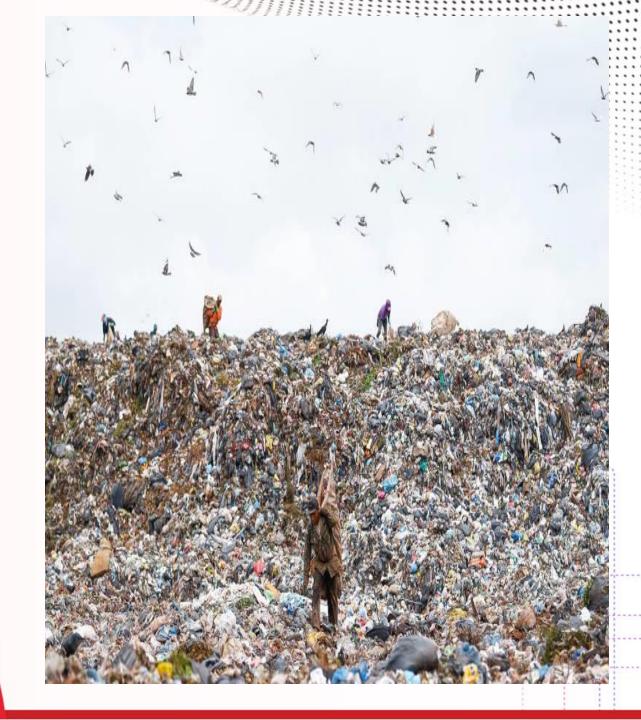


```
public
  procedure emitirNotaFiscal(ANumeroNota: Integer);
  procedure Cancelar_Nota_Fiscal ( numero_nota: integer );
  procedure enviar_nota_por_email (numero_nota : integer);
```

```
public
  procedure EmitirNotaFiscal(ANumeroNota: Integer);
  procedure CancelarNotaFiscal(ANumeroNota: Integer);
  procedure EnviarNotaPorEmail(ANumeroNota: Integer);
```

```
item: The EntityVendaItem;
 PedidoItem: TRP: LityPedidoItem;
 LDAO: Tive VendaItem;
begin
for PedidoItem in FPedido.itens do
begin
     if PedidoItem.fracoes.Count > 0 then begin
       Continue;
     end;
   if item.tamanho = EmptyStr then
     BEGIN
     item.vendaPorTamanho := False;
     item.tamanho := 'M';
   END;
   FVenda.itens.Add(item
item.idVenda := FVenda.id;
item.idEmpresa := FVenda.idEmpresa;
item.produto := PedidoItem.produto;
   item.numeroItem := FVenda.itens.Count;
     PedidoItem.numeroItem := item.numeroItem;
     item.quantidade := PedidoItem.quantidade;
     item.valorUnitario := PedidoItem.valorUnitario:
     item.valorTotalProduto := item.quantidade * item.valorUnitario;
   item.tamanho := PedidoItem.tamanho;
   item.numeroItemFracionado := 0;
   item.vendaPorTamanho := True;
   item.observacao := PedidoItem.observacao;
   if
         PedidoItem.fracoes.Count > 0 then
   begin
```

```
item: T EntityVendaItem;
  PedidoItem: TRP=====ntityPedidoItem;
  LDAO: Till OVendaItem;
begin
for PedidoItem in FPedido.itens do
begin
      if PedidoItem.fracoes.Count > 0 then begin
        SalvarItemr. ( PedidoItem);
       Continue:
      end;
    if item.tamanho = EmptyStr then
      BEGIN
      item.vendaPorTamanho := False;
      item.tamanho := 'M';
    END;
    item := Transfer ityVendaItem.Create;
FVenda.itens.Add(item ) ;
item.idVenda := FVenda.id;
item.idEmpresa := FVenda.idEmpresa;
item.produto := PedidoItem.produto;
    item.numeroItem := FVenda.itens.Count;
      PedidoItem.numeroItem := item.numeroItem;
      item.guantidade := PedidoItem.guantidade;
      item.valorUnitario := PedidoItem.valorUnitario;
      item.valorTotalProduto := item.quantidade * item.valorUnitario;
    item.tamanho := PedidoItem.tamanho;
    item.numeroItemFracionado := 0;
    item.vendaPorTamanho := True;
    item.observacao := PedidoItem.observacao;
         PedidoItem.fracoes.Count > 0 then
    if
    begin
```



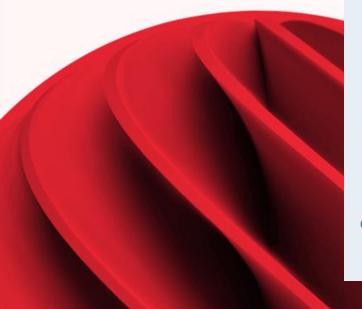
```
var
  LCount: Integer;
  LItem: TNFCeModeloItemNota;
begin
  FNota := ANotaFiscal;
  LItem := nil;
  for LCount := 0 to Pred(FNota.Itens.Count) do
  try
    LItem := FNota.Itens[LCount];
    Aplicar(LItem);
  except
    on E: Exception do
    begin
      E.Message := Format('Item %d - %s %s: %s', [(LCount + 1),
        LItem.CodigoProduto, LItem.DescricaoProduto, E.Message]);
      raise;
    end;
  end;
  ANotaFiscal.IcmsTotais.AplicarValores;
  ANotaFiscal.IssqnTotais.AplicarValores;
```





https://docwiki.embarcadero.com/RADStudio/Alexandria/en/Delphi%E2%80%99s Object Pascal Style Guide







PAGE

DISCUSSION

RAD Studio 11 Alexandria

Topics

Libraries Reference

Code Examples

RAD Studio 11 Topics

What's New

Tutorials

FireMonkey Application Platform

Multi-Device Applications

Getting Started

Steps in Developing a Project

Key Application Types

FireDAC

Windows Developer's Guide

Modeling Tools

IDE Reference and Utilities

Delphi Reference

C++ Reference

Subject Index

Other Versions

Latest version

Sydney Topics

Delphi's Object Pascal Style Guide

Go Up to Delphi Language Reference

This document is an updated version of the classic Object Pascal style guide, originally curated by Charlie Calvert for Borland a Delphi Product Manager Marco Cantu, with contributions from the internal R&D team, Delphi MVPs, and the Delphi communit

11.3 release available - Learn More! ര

Index

- Introduction
- General Rules: Identifiers, Keywords, Indentation
- · Source Code Files, Units, and Their Structure
- White Space Usage
- Comments
- Statements
- Type Declarations

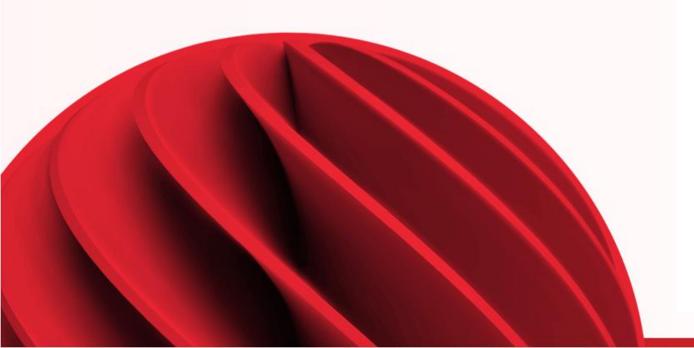
Next

Introduction

Category: Delphi

Vantagens

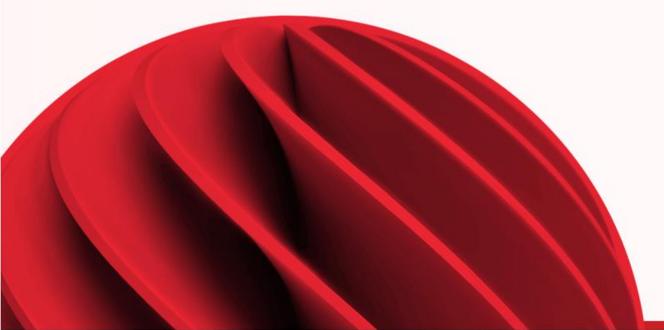
- Leitura fácil e amigável do código.
- Longevidade do projeto.
- Facilidade pra receber novos devs.
- Capacita o dev a integrar em outros projetos e empresas de grande porte.
- Evolução do dev junto com a linguagem.





Desvantagens

- Você vai se tornar um cara chato.





Que seu código já seja a documentação

```
public
  // Esse método vai emitir a nota fiscal NFe
  procedure Executar;
```

```
public
   procedure EmitirNFe;
```

Evite abreviações

```
procedure TForm1.EmitirNFe;

var

DescNota: Variant;

begin

end;
```

```
procedure TForm1.EmitirNFe;
var
StringDescontoNota: string;
begin
end;
```

```
procedure TForml.EmitirNFe;
var
DescontoNota: Variant;
begin
end;
```

```
procedure TForm1.EmitirNFe;
var
StrDescontoNota: string;
begin
end;
```

Palavras chave minúsculas

```
∃unit Unit3;
∃interface
 uses
   Winapi.Windows, Winapi.Messages, System.
   Vcl.Controls, Vcl.Forms, Vcl.Dialogs;
 type
  TForm3 = class(TForm)
  public
    procedure EmitirNFe;
   end;
 var
   Form3: TForm3;
procedure Register;
```

```
implementation
{$R *.dfm}
procedure Register;
begin
end;
 { TForm3 }
procedure TForm3.EmitirNFe;
var
  LDescricao: string;
  LNumero: Integer;
begin
end;
end.
```

Declarações – Variáveis Locais

- As variáveis locais devem ter **Camel Caps** como todos os outros identificadores.
- Considere que nomes de campo de 1 caractere devem ser evitados, exceto para variáveis temporárias e de loop.
- Alguns desenvolvedores definem suas próprias regras como um L inicial, o que faz sentido, mas não temos uma recomendação estrita para um prefixo de variáveis locais.

```
// Correto
procedure TForm3.EmitirNFe;
var
   I: Integer;
   LEmitente: string;
   LValorNota: Double;
begin
   for I := 0 to 10 do
```

```
// Incorreto
procedure TForm3.EmitirNFe;
var
   i: Integer;
   emitente: string;
   Valor_Nota: Double;
begin
   for i := 0 to 10 do
```

Declarações – Variáveis Locais

- Evite declarar vários identificadores em uma única linha mesmo sendo do mesmo tipo.
- Utilizar 1 único espaço após os dois pontos após o nome da variável.
- Evitar o alinhamento vertical dos nomes das variáveis.

```
// Correto
procedure TForm1.EmitirNFe;
var
    I: Integer;
    LEmitente: string;
    LDestinatario: string;
    LValor: Double;
begin
    for I := 0 to 10 do
```

Declarações – Variáveis Locais

Recomendamos o uso de variáveis **inline** para declarações de escopo local de bloco (ou seja, não declare no método var variáveis de bloco que são usadas apenas em um bloco específico). Um exemplo é uma variável local de loop for, que **recomendamos** declarar usando variáveis inline:

```
// Clássico, mas agora considerado incorreto
procedure TForml.EmitirNFe;
var
   I: Integer;
   LEmitente: string;
   LDestinatario: string;
   LValor: Double;
begin
   for I := 0 to 10 do
```

```
// Correto
procedure TForm1.EmitirNFe;
var
   LEmitente: string;
   LDestinatario: string;
   LValor: Double;
begin
   for var I := 0 to 10 do
```

Declarações – if

- As instruções if devem sempre aparecer em pelo menos 2 linhas.
- Não adicione parênteses extras ao redor de uma instrução.
- Cada instrução de separação (begin, end, else) em linha separada, ou seja, não coloque begin após o then.

```
// INCORRECT
if A < B then DoSomething;

// INCORRECT
if (A < B) then
   DoSomething;

// CORRECT
if A < B then
   DoSomething;</pre>
```

```
// INCORRECT
if A < B then begin
  DoSomething;
  DoSomethingElse;
end else begin
  DoThis;
  DoThat;
end;</pre>
```

```
// CORRECT
if A < B then
begin
    DoSomething;
    DoSomethingElse;
end
else
begin
    DoThis;
    DoThat;
end;</pre>
```

Declarações – if

- Algumas variações são consideradas válidas. Um deles usa uma instrução **if** na mesma linha de uma condição **else**, mantendo a indentação original do bloco **if** externo (enquanto formalmente este seria um bloco aninhado com indentação adicional).

```
if Condition then
begin
   DoThis;
end
else if Condition then
begin
   DoThat;
end;
```

Procedures e Functions

- Se possível, uma procedure ou function deve aparecer em uma linha.
- Não coloque parênteses na declaração de procedures sem parâmetros.

```
// Correto
function SalvarArquivo(Conteudo, Caminho: string): Boolean;
// Correto
function SalvarArquivo(Conteudo: string; Caminho: string): Boolean;
// Incorreto
function SalvarArquivo (Conteudo, Caminho: string): Boolean;
```

```
// Correto
procedure EmitirNFe;
// Incorreto
procedure EmitirNFe();
```

Procedures e Functions

- As linhas seguintes devem ser alinhadas com 2 espaços a esquerda.
- Não é recomendado colocar cada parâmetro em uma linha separada.

Parâmetros

Alguns desenvolvedores definem suas próprias regras como um **A** inicial, que significa "**Argumento**" e é frequentemente usado no código-fonte das bibliotecas Delphi - mas não temos uma recomendação estrita para um prefixo de nome de parâmetro.

```
// Correto
function SalvarArquivo(Conteudo, CaminhoArquivo: string): Boolean;
// Correto
function SalvarArquivo(AConteudo: string; ACaminhoArquivo: string): Boolean;
// Incorreto
function SalvarArquivo(conteudo, caminho_arquivo: string): Boolean;
```

Declaração de uses

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics, Vcl.Controls, Vcl.Forms, Vcl.Dialogs;

```
Winapi.Windows,
Winapi.Messages,
System.SysUtils,
System.Variants,
System.Classes,
Vcl.Graphics,
Vcl.Forms,
Vcl.Forms,
Vcl.Dialogs;
```

Declaração de uses - EVITE

```
uses Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,
    System.Classes, Vcl.Graphics, Vcl.Controls, Vcl.Forms, Vcl.Dialogs;
```

```
uses
Winapi.Windows,
Winapi.Messages,
System.SysUtils,
System.Variants,
System.Classes,
Vcl.Graphics,
Vcl.Graphics,
Vcl.Forms,
Vcl.Dialogs;
```

Linhas em branco – Onde usar?

- Entre declarações de classe.
- Entre as implementações do método.
- Às vezes, dentro de um método para separar áreas lógicas de código, como antes de um loop.

```
TForm1 = class(TForm)
  public
    procedure EmitirNFe;
    procedure CancelarNFe;
  end:
  TMinhaClasse = class
  public
  end;
  Form1: TForm1;
implementation
{$R *.dfm}
procedure TForm1.CancelarNFe;
begin
end;
procedure TForm1.EmitirNFe;
  DescontoNota: Double;
  NumeroNota: Integer;
begin
  NumeroNota := 5;
  for I := 0 to 5 do
  begin
  end:
end;
```

Linhas em branco – Onde NÃO usar?

- Declaração type.
- Separando a declaração do método do corpo
- Depois de outra linha em branco

```
type
  TForm1 = class(TForm)
  public
    procedure EmitirNFe;
    procedure CancelarNFe;
  end;
  TMinhaClasse = class
  public
  end;
implementation
{$R *.dfm}
procedure TForm1.EmitirNFe;
var
  DescontoNota: Double;
  NumeroNota: Integer;
begin
  NumeroNota := 5;
  for I := 0 to 5 do
```

Espaços em branco – Onde usar?

- Em torno de operadores.
- Após uma vírgula separando os parâmetros em uma chamada de função ou método.
- Após uma vírgula separando tipos genéricos.
- Após o ponto e vírgula na declaração dos parâmetros de uma função ou método.

```
begin
   // Correto
   TotalNota := TotalProdutos + TotalJuros;
   // Incorreto
   TotalNota := TotalProdutos+TotalDesconto;
end;
```

```
begin
  // Correto
  Descricao := Descricao.Replace(',', '.');
  // Incorreto
  Descricao := Descricao.Replace(',','.');
end;
```

```
var
  // Correto
  Dicionario: TDictionary<string, string>;
  // Incorreto
  Dicionario: TDictionary<string,string>;
```

```
// Correto
function SalvarArquivo(Conteudo, Caminho: string): Boolean;
// Incorreto
function SalvarArquivo(Conteudo, Caminho: string):Boolean;
```

Espaços em branco – Onde NÃO usar?

- Entre um nome de método e seu parêntese de abertura
- Após um parêntese de abertura ou antes de um parêntese de fechamento
- Após um colchete de abertura ou antes de um colchete de fechamento
- Antes dos dois pontos para uma declaração de variável; em outras palavras, nenhum espaço em branco entre uma variável e o :

```
// Incorreto
procedure TForm1.EmitirNFe;
var
   DescontoNotaFiscal : Double;
   TotalNota : Double;
   Descricao : string;
begin
   Descricao := Format ( 'teste %s.' , [ Descricao ] );
```

```
// Correto
procedure TForm1.EmitirNFe;
var
   DescontoNotaFiscal: Double;
   TotalNota: Double;
   Descricao: string;
begin
   Descricao := Format('teste %s.', [Descricao]);
```

Espaços em branco – Onde NÃO usar?

```
procedure TForm1.EmitirNFe;
var

DescontoNotaFiscal: Double;
TotalNota : Double;
Descricao : string;

begin
DescontoNotaFiscal := 0;
TotalNota := 100;
Descricao := 'Teste';
end;
```

```
procedure TForm1.EmitirNFe;
var
   DescontoNotaFiscal: Double;
   TotalNota: Double;
   Descricao: string;
begin
   DescontoNotaFiscal := 0;
   TotalNota := 100;
   Descricao := 'Teste';
end;
```

Linhas de continuação

// Incorreto

then begin

(ValorJuros > 0)

if (Emitente <> Destinatario) or (ValorDesconto > 0) or

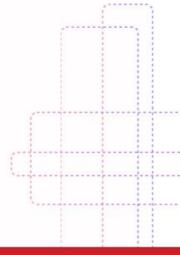
Types

Como regra geral, todos os identificadores de nome de tipo são prefixados com um T maiúsculo, como em *TMyType*.

```
// Correto
TCliente = class
end;

// Incorreto
Cliente = class
end;

// Incorreto
ECliente = class
end;
```



Types - Ressalvas

- Classes de exceção (classe Exception e filhas) iniciam com letra E (EMyException).
- Os tipos de interface são prefixados com a letra maiúscula I (IMyInterface).
- Classes de atributos personalizados (descendentes de **TCustomAttribute**), não usam a inicial T e devem terminar com a palavra Attribute (veremos adiante).

```
type
  // Correto
  EMinhaException = class(Exception)
  end;

// Incorreto
  TMinhaException = class(Exception)
  end;

// Incorreto
  MinhaException = class(Exception)
  end;
```

```
type
  // Correto
  IMinhaInterface = interface
      ['{D5964FEA-184E-4293-8B7A-9C6DD8441757}']
  end;

end;

// Incorreto
  TMinhaInterface = interface
      ['{F4B27EC3-BFAE-40E9-B013-991DFFE798DE}']
  end;

// Incorreto
  MinhaInterface = interface
      ['{3908A35D-7980-437B-8967-22441E05524A}']
  end;
```

Corpo da Classe

O corpo de uma declaração de classe deve ser organizado na seguinte ordem, para cada um dos blocos:

- Fields
- Procedures e Functions
- Properties

```
// Correto
TForm1 = class(TForm)
private
   FNome: string;
public
   procedure EmitirNFe;
   procedure CancelarNFe;

property Nome: string read FNome write FNome;
end;
```

```
// Incorreto
TForm1 = class(TForm)
private
  FNome: string;
public
  property Nome: string read FNome write FNome;
  procedure EmitirNFe;
  procedure CancelarNFe;
end;
```

Corpo da Classe

```
TForm1 = class(TForm)
private
  FNome: string;
public
  // Normalmente no início
  constructor Create;
  destructor Destroy; override;
  procedure EmitirNFe;
  procedure CancelarNFe;
  property Nome: string read FNome write FNome;
end;
```

```
TForm1 = class(TForm)
private
  FNome: string;
public
  // Incomum
  procedure EmitirNFe;
  procedure CancelarNFe;
  constructor Create;
  destructor Destroy; override;
  property Nome: string read FNome write FNome;
end;
```

Nomenclatura dos campos

- Campos (Fields) começam cada declaração de tipo com F maiúsculo.
- Propriedades (property) não usam prefixos.

```
TForm1 = class(TForm)
private
  FNome: string;
  //Correto
  FSobrenome: string;
  // Incorreto
  Sobrenome: string;
public
  // Incorreto
  Idade: Integer;
  // Correto
  FIdade: Integer;
  property Nome: string read FNome write FNome;
  // Correto
  property DataNascimento: TDateTime;
  // Incorreto
  property FDataNascimento: TDateTime;
```

Property – EVITE GET e SET desnecessário

```
type
  TCliente = class
  private
     FNome: string;
    procedure SetNome(const Value: string);
    function GetNome: string;
  public
    property Nome: string read GetNome write SetNome;
  end;
implementation
 { TCliente }
function TCliente.GetNome: string;
begin
  Result := FNome;
end:
procedure TCliente.SetNome(const Value: string);
begin
  FNome := Value;
end;
```

```
type
 TCliente = class
 private
    FNome: string;
 public
    property Nome: string read FNome write FNome;
  end;
implementation
end.
```

Property – EVITE GET e SET desnecessário

```
type
  TAluno = class
  private
    FNome: string;
    FNotal: Double;
    FNota2: Double;
    function GetMedia: Double;
    procedure SetNome(const AValue: string);
  public
    property Nome: string read FNome write SetNome;
    property Notal: Double read FNotal write FNotal;
    property Nota2: Double read FNota2 write FNota2;
    property Media: Double read GetMedia;
  end;
implementation
function TAluno.GetMedia: Double;
begin
  Result := (Nota1 + Nota2) / 2;
end;
procedure TAluno.SetNome(const AValue: string);
begin
  FNome := UpperCase(AValue);
end;
```

Isso não é tudo.

https://docwiki.embarcadero.com/RADStudio/Alexandria/en/Delphi%E2%80%99s Object Pascal Style Guide





OBRIGADO!!!!!!

O que você achou da palestra?

Acesse o link do QR Code ao lado e responda a pesquisa.

