Padrões de código

Documento para consulta e aprendizado de padrões de códigos e boas práticas.

Índice

Resumo

O livro "Padrões de Código", contém todas as informações referentes à padronização e boas práticas utilizadas na implementação e desenvolvimento dos sistemas SS. Organizado por capítulos e sempre atualizado, este livro é de suma importância para a equipe de desenvolvimento, pois apresenta as diretrizes à serem seguidas durante a manutenção do código. Logo abaixo, serão apresentados os capítulos e uma breve descrição do que é tratado em suas respectivas páginas.

Sumário

Exemplo: DELETE

Exemplo de cláusula de DELETE.

Exemplo: CASE

Exemplos de cláusulas com CASE, tanto com expressão quanto sem expressão no CASE.

Palavras reservadas

Boas Práticas

Versionamento

Definição das diretrizes para mensagens de versionamento. Mensagem modelo e exemplo de mensagem em criação de branch, commit de alterações e atualização de branch.

Documentação Interna

Exemplos de comentários e como devem ser utilizados.

Código Limpo

Definição de código limpo e *bad smells.* Exemplos de *bad smells* comuns e possíveis correções.

Delphi

Capítulo destinado aos padrões como nomenclatura, indentação entre outros, usados em Delphi.

Nomenclatura Geral

Nomenclatura

Componentes

Os componentes possuem seus prefixos utilizando letras que remetem ao nome completo da classe do componente. Componentes herdados dos componentes listados e sem conexão ao banco de dados devem possuir o mesmo prefixo do componente principal, com exceção dos componentes visuais com conexão ao banco de dados, que utilizam o prefixo “DB“ seguido do padrão identificado logo abaixo. Exemplo: DBED_Nome é um componente da classe TDBEdit, responsável pela representação de um nome.

Seguem alguns exemplos de prefixos padrões utilizados:

ED_: TEdit;	TS_: TTabSheet;
PN_: TPanel;	RB_: TRadioButton;
GB_: TGroupBox;	OD_: TOpenDialog;
LB_: TLabel;	SD_: TSaveDialog;
MM_: TMemo;	Q_: TQuery, TFDQuery;
BT_: TButton (Botões num geral);	DS_: TDataSource;
CK_: TCheckBox;	CDS_: TClientDataSet;
CB_: TComboBox;	DSP_: TDataSetProvider;
LT_: TListBox;	T_: TTable;
RG_: TRadioGroup;	FR_: TFrxReport;
PC_: TPageControl;	FDS_: TFrxDBDataSet;
TB_: TToolBar;	RV_: TRvProject;

GD_: TGrid;	DSC_: TRvDataSetConnection;
MI_: TMenuItem;	SH_: TShape
IM_: TImage	TM_: TTimer

Exemplo de componentes visuais com conexão ao banco de dados:

DBED_: TDBEdit;	DBGD_: TDBGrid;
DBCK_: TDBCheckBox;	DBRG_: TDRadioGroup;

Variáveis

O nome da variável deve descrever de maneira clara e sem abreviações a função da variável. Utilizar a notação PascalCase para o nome da variável.
As variáveis podem ter ainda um prefixo para adicionar mais informações sobre sua origem. Para **variáveis locais** utiliza-se o prefixo **‘L’**, para **argumentos (Parâmetros) de métodos** utiliza-se o **'A’**. Para **atributos (Field)** de classes utiliza-se o **‘F’**.

```
type
  TPessoa = class
    protected
      FNome: String; // Campo nome da classe Pessoa. Prefixo F
    public
      constructor Create(ANome: String); // Argumento do método. Prefixo A.
  end;

implementation

constructor TPessoa.Create(ANome: String);
var
  LNome: String; // Variável local. Prefixo L
begin

end;
```

Constantes

Para nomes de constantes, o padrão utilizado é:

```
const
  NOME_DA_CONSTANTE_1 = 0;
```

```
  NOME_DA_CONSTANTE_2 = False;
  NOME_DA_CONSTANTE_3 = 'Teste';
```

O tipo da constante é deduzido através do valor dessa constante no momento da declaração. Porém, é possível declarar explicitamente um tipo para as constantes, e em alguns casos, como o de arrays, é necessário que esse tipo seja declarado.

```
const
  NOME_DA_CONSTANTE_4: Double = 10;
  NOME_DA_CONSTANTE_5: array [1..2] of String = ('a', 'b');
```

Métodos

Assim como o nome das variáveis, o nome do método deve seguir o mesmo modelo, com nomes que descrevem a função do método, utilizando também a notação PascalCase.
Ao editar um método, é importante manter sua função original, caso sua funcionalidade seja alterada, deve-se refatorar o seu nome para que permaneça fiel ao seu comportamento.

Exemplo 1: Criar um método que faça a confirmação de um rotina de gravação. Sem retorno e sem argumentos.

```
procedure ConfirmarGravacao();
```

Exemplo 2: Criar um método que faça o cálculo da área de um retângulo. Seu retorno será um inteiro, esse método terá ainda dois argumentos, altura e largura.

```
function CalcularArea(AAltura, ALargura: Integer): Integer;
```

Units

As units padrões utilizadas anteriormente para as entidades eram nomeadas acrescentando 1, 2, 3 ou 4. Para facilitar o entendimento na leitura do nome da unit, elas passam a ser utilizadas no singular, seguido do nome de sua funcionalidade. Por exemplo, a entidade "Empresa" passa a ser utilizada como:
● Empresas1: Empresa.Manutencao;
● Empresas2: Empresa.Cadastro;
● Empresas3: Empresa.Pesquisa;
● Empresas4: Empresa.DataModule;
Para units que tenham outra função que não se enquadrem nessas descritas anteriormente, o padrão é utilizar o nome da entidade seguido da descrição de sua função.
Exemplo: Empresa.Copiar.

Classes

O padrão Delphi para nome de classes aconselha que ela inicie com o prefixo T (*Type*). É aconselhável que o nome dessas classes sejam descritivos quanto à sua entidade, função e/ou tipo. Por exemplo, uma classe que define a entidade produtos poderia ter o nome *TProdutoModelo*, enquanto o formulário para importação desses produtos poderia ser *TProdutoImportacaoFormulario.*

Espaçamento

Regras para Formatação Horizontal

Tabulação: Utilizar 2 caracteres (espaços) ao invés da tabulação.

Para manter a estrutura do código de modo que sua leitura seja fácil, o número máximo de caracteres por linha é **130**. Caso a expressão ultrapasse esse comprimento, a linha deve ser quebrada segundo essas regras:

1. A linha tem comparações? `('a = b', 'a <> b', 'a <= b', 'a >= b', 'a < b' ou 'a > b')`
    1. Envolver cada comparação com parênteses.
2. A linha contem negação? `('not')`
    1. Envolver cada negação com um parênteses.
3. A linha contém operadores lógicos? `('and' e 'or')`
    1. Envolver cada sequência de `'and'` com um parênteses;
    2. Envolver cada sequência de `'or'` com um parênteses.
4. A linha é uma atribuição? `(':=')`
    1. Adicionar uma quebra de linha após o sinal de atribuição `(':=')`, alinhar a nova linha com a linha original e indentar em um nível (2 espaços);
    2. Caso o código resultante não tenha ultrapassado a linha das 130 colunas, processo pode ser interrompido;
    3. Caso a atribuição seja uma String, ela pode ser quebrada em uma ou mais concatenações. Essas podem ser quebradas e alinhadas com a linha original, adicionando um nível de indentação;
    4. Caso a atribuição seja uma comparação (Não considerar comparações dentro de métodos aninhados), adicionar uma quebra de linha após o sinal de comparação `('=', '<>', '<=', '>=', '<' ou '>')`, e alinhar a nova linha com o primeiro operando.
5. A linha tem um 'if'?
    1. Caso a condição do if não tenha um parênteses envolvendo a condição inteira, envolvê-la com um;
    2. Adicionar uma quebra de linha após cada operador `'and'` ou `'or'`;
    3. Após cada quebra de linha, alinhar a nova linha com o parênteses onde ela está contida e adicionar um espaço;
    4. Caso o código resultante não tenha ultrapassado a linha das 1 colunas, processo pode ser interrompido;
    5. Caso alguma das condições do if tenha uma comparação (Não considerar comparações dentro de métodos aninhados), adicionar uma quebra de linha após o sinal de comparação `('=', '<>', '<=', '>=', '<'` ou `'>')`, e alinhar a nova linha com o primeiro operando.

    6. Caso o código resultante não tenha ultrapassado a linha das 130 colunas, processo pode ser interrompido.
6. A linha contém métodos aninhados?
    1. Aplicar essa regra do métodos mais externo para o mais interno;
    2. Adicionar uma quebra de linha exatamente antes do início do método aninhado;
    3. Alinhar a nova linha com o método onde ela está contida (Caso o método tenha uma negação (not), alinhar com o início dessa negação), adicionar ainda um nível de indentação (2 espaços);
    4. Caso o código resultante não tenha ultrapassado a linha das 130 colunas, processo pode ser interrompido.
7. A linha contém parâmetros?
    1. Ao realizar a quebra de linha de um parâmetro, ele deve estar indentado um nível (2 espaços) em relação ao nível do método proprietário do parâmetro (caso o método proprietário tenha uma negação (not), alinhar com o início dessa negação);
    2. Caso o código resultante não tenha ultrapassado a linha das 130 colunas, processo pode ser interrompido.
8. Após todas essas etapas a linha ainda ultrapassa a marca de 130 colunas?
    1. Caso ainda assim, a linha ultrapasse a marca de 130 colunas então deve-se verificar a existência de pontos na expressão (“.”), sendo que, ao realizar esse tipo de quebra de linha, o ponto deve ficar com a expressão na linha inferior, indentado em um nível em relação ao início da expressão.

Observação: Algumas vezes, quando uma linha ultrapassa as 130 colunas, é um sinal de que esse trecho pode ser refatorado. O programador pode então refatorar, se julgar que é necessário ou melhor.

Declaração de variáveis

> **Correto**

```
var
  LTeste: Integer;
```

> **Incorreto**

```
var
  LTeste : Integer;
  LTeste :Integer;
```

Atribuições

> **Correto**

```
LTeste := 15;
```

Incorreto

```
LTeste:=15;
LTeste:= 15;
LTeste :=15;
```

Métodos

Correto

```
procedure Ex(AParametro: Integer);
procedure Ex(AParametro1, AParametro2: Integer);
procedure Ex(AParametro: Integer; AParametro2: String);
```

Incorreto

```
procedure Ex (AParametro1: Integer);
procedure Ex( AParametro: Integer);
procedure Ex(AParametro: Integer );
procedure Ex(AParametro1,AParametro2: Integer);
procedure Ex(AParametro1 , AParametro2: Integer);
procedure Ex(AParametro: Integer;AParametro2: String);
procedure Ex(AParametro: Integer ; AParametro2: String);
```

Matrizes

Correto

```
LTeste := LMatriz[0];
```

Incorreto

```
LTeste := LMatriz[ 0];
LTeste := LMatriz[0 ];
LTeste := LMatriz[ 0 ];
```

Operadores binários

Correto

```
LTeste := 1 + 1;
```

Incorreto

```
LTeste := 1+1;
LTeste := 1 +1;
LTeste := 1+ 1;
```

Operadores unários

Correto

```
LTeste := -1;
```

Incorreto

```
LTeste := - 1;
LTeste :=-1;
```

Subrotinas

Correto

```
function MeuMetodo: String;

    procedure SubMetodo;
    begin
      //Código SubMetodo
    end;

begin
  //Código MeuMetodo
end;
```

Incorreto

```
function MeuMetodo: String;

procedure SubMetodo;
begin
  //Código SubMetodo
end;
begin
  //Código MeuMetodo
end;
```

```
function MeuMetodo: String;

  procedure SubMetodo;
  begin
    //Código SubMetodo
  end;

begin
  //Código MeuMetodo
end;
```

Uses

> **Dica:** Para identificar se uma unit deve ser declarada na uses superior ou inferior, realize o seguinte teste: Copie a unit para a uses inferior e compile o programa, caso o processo de compilação não apresente erros, a unit deve permanecer na uses inferior. Caso contrário, mova a unit para a uses superior.

No primeiro “uses“ (logo abaixo de “interface”) declarar as units no .dfm e nas assinaturas dos métodos.

```
interface

uses
  Winapi.Windows,
  Winapi.Messages,
  System.SysUtils,
  System.Classes,
```

```
  Vcl.Graphics,
  Vcl.Controls,
  Vcl.Forms,
  Vcl.Dialogs;
```

No segundo “uses“ (logo abaixo de “implementation”) declarar as units usadas na implementação do código.

```
implementation

uses
  uExisteTabela,
  uProcuraRegistro,
  uGravaSistema;
```

A fim de evitar conflito em Merge devido a alterações em mesma linha, cada Unit deve ser declarada em linha diferente. Esse erro costuma acontecer principalmente quando há refatoração ou exclusão de algum componente visual.

> Correto

```
implementation

uses
  uExisteTabela,
  uProcuraRegistro,
  uGravaSistema;
```

> Incorreto

```
implementation

uses
  uExisteTabela, uProcuraRegistro, uGravaSistema;
```

Arrays

Ao indentar elementos de um array, eles devem seguir a lógica de ordenação de parênteses. Por não estarem em níveis diferentes de indentação, devem ser indentados da seguinte forma

>

Correto

```
TClasse.Metodo(
  Parametro1,
  Parametro2,
  [Item1, Item2, Item3
   Item4, Item5, Item6]);
```

Incorreto

```
TClasse.Metodo(
  Parametro1,
  Parametro2,
  [Item1, Item2, Item3
    Item4, Item5, Item6]);
```

```
TClasse.Metodo(
  Parametro1,
  Parametro2,
  [Item1, Item2, Item3
  Item4, Item5, Item6]);
```

IN em tratamento de condições

Correto

```
Result :=
  (Self in
    [cdfIncidenciadecisaojudicial,
     cdfIncidenciaDecisaoJudicial13Sal,
     cdfIncidenciaDecisaoJudicialAvisoPrevioIndenizado]);
```

Incorreto

```
Result :=
  Self in [cdfIncidenciadecisaojudicial,
           cdfIncidenciaDecisaoJudicial13Sal,
```

```
        cdfIncidenciaDecisaoJudicialAvisoPrevioIndenizado];
```

```
Result := (Self in [cdfIncidenciadecisaojudicial,
                    cdfIncidenciaDecisaoJudicial13Sal,
                    cdfIncidenciaDecisaoJudicialAvisoPrevioIndenizado]);
```

Palavras reservadas

As palavras reservadas da linguagem Delphi devem ser escritas com todas as letras minúsculas, a única exceção é a palavra String, pois segue o padrão dos tipos, os quais iniciam com letra maiúscula. As palavras reservadas do Delphi são:

dispid	near	then
dispinterface	nil	threadvar
div	not	to
do	object	try
downto	of	type
dynamic	operator	unit
else	or	until
end	out	uses
except	overload	var
export	override	virtual
exports	package	while
external	pascal	write
far	private	writeonly
file	procedure	xor
finally	program	
for	property	

SQL

Neste capítulo se encontram os padrões de código adotados para SQL e alguns exemplos ilustrativos, caso mesmo com os exemplos descritos, ainda restem dúvidas, não hesite em perguntar

Considerações Iniciais

Introdução

Para rotinas em SQL, deve-se observar as seguintes considerações:

1. **Palavras reservadas** devem ser escritas todas com as **letras maiúsculas**.
2. Nomes de **tabelas e campos** devem utilizar notação **PascalCase**.
3. **Parâmetros** devem possuir o **prefixo "P"** e utilizar notação **PascalCase**.
4. Para facilitar a implementação e leitura de cláusulas, deve **existir uma coluna do início ao fim da cláusula em que é feita a separação de palavras reservadas e campos/tabelas**, como no exemplo abaixo:

```
    SELECT SUM(T1.Campo1) Campo1,
           T2.Campo1
      FROM Tabela1 T1
INNER JOIN Tabela T2
        ON T2.Campo2 = T1.Campo1
     WHERE T1.Campo3 = :PTabela1Campo3
       AND T2.Campo3 = :PTabela2Campo3
  GROUP BY T2.Campo1
 UNION ALL
    SELECT (SELECT Campo 1
              FROM Tabela 3) Campo1,
           T2.Campo1
      FROM Tabela4 T4
INNER JOIN (SELECT Campo1
              FROM Tabela5) T5
        ON T5.Campo1 = T4.Campo1
     WHERE T4.Campo3 = :PTabela4Campo3
  ORDER BY 1
```

Perceba que na cláusula exemplo acima, as linhas 4 e 14 possuem **as maiores palavras reservadas/expressões da cláusula**, logo **todas as outras palavras reservadas recebem a adição de pelo menos um espaço em branco à sua esquerda**, para que **o fim de todas as expressões finalizem na mesma coluna**. Formando assim, uma **coluna do início ao fim da cláusula que divide palavras reservadas para a esquerda e campos/tabelas para a**

direita.

SQL

Exemplo: CASE

Sem expressão no CASE

```
SELECT T1.Campo1, T1.Campo2
  FROM Tabela1 T1
 WHERE T1.Campo2 = :PInformacao1
   AND T1.Campo3 = :PInformacao2
   AND CASE
         WHEN
           T1.Campo4 IS NOT NULL
         THEN
           T1.Campo4 = :PInformacao3
         ELSE
           T1.Campo5 = :PInformacao3
       END
```

Com expressão no CASE

```
SELECT T1.Campo1, T1.Campo2
  FROM Tabela1 T1
 WHERE T1.Campo2 = :PInformacao1
   AND T1.Campo3 = :PInformacao2
   AND CASE COALESCE(T1.Campo4, "")
         WHEN
           ""
         THEN
           T1.Campo5 ||
           SUBSTRING(T1.Campo6 FROM 1 FOR 1) ||
           SUBSTRING(T1.Campo6 FROM 3 FOR 5)
         ELSE
           T1.Campo7
       END ApelidoDoCampo
```

Com CASE no SELECT

```
SELECT
  CASE
    WHEN
      EXISTS (SELECT 1
        FROM Tabela1
          WHERE Campo1 = :PCampo1
            AND Campo2 = :PCampo2)
      AND EXISTS (SELECT 1
                  FROM Tabela1
                 WHERE Campo1 = :PCampo1
                   AND Campo2 = :PCampo2)
    THEN
    "True"
    ELSE
    "False"
  END AS Resultado
  FROM RDB$DATABASE;
```

SQL

Exemplo: DELETE

```
DELETE
  FROM TabelaExemplo
 WHERE Abrev = :PAbrev
   AND Tipo = :PTipo
   AND Categoria = :PCategoria
```

> As comparações realizadas nas linhas 3-5 poderiam ter seus operadores alinhados na mesma coluna, entretanto essa não é uma prática recomendada para cláusulas mais extensas.

SQL

Exemplo: Modificador DISTINCT

```
SELECT DISTINCT T.Campo1, T.Campo2, T.Campo3
                T.Campo4, T.Campo5, T.Campo6
           FROM Tabela T
     INNER JOIN OutraTabela O
             ON O.Campo1 = T.Campo7
          WHERE T.Campo1 = :Param1
            AND T.Campo8 = :Param2
```

SQL

Exemplo: UPDATE

```
UPDATE TabelaExemplo
  SET Numero = :PNumero,
      Nome = :PNome
 WHERE Categoria = :PCategoria
  AND Tipo = :PTipo
```

> As comparações realizadas nas linhas 2-5 poderiam ter seus operadores alinhados na mesma coluna, entretanto essa não é uma prática recomendada para cláusulas mais extensas

SQL

Palavras Reservadas

As palavras reservadas em SQL devem ser escritas com todas as letras maiúsculas:

ACTIVE	DOUBLE	NUMERIC
ADD	DROP	OF
AND	EDIT	ON
ALL	ELSE	ONLY
ALTER	END	OPEN
ANY	EXECUTE	OR
AS	EXISTS	ORDER
ASC	FILTER	OUTER
ASCENDING	FIRST	PERCENT
AT	FLOAT	PLAN
AVG	FOR	POSITION
BEFORE	FOREIGN	PRECISION
BEGIN	FROM	PREPARE
BETWEEN	FULL	PRIMARY
BLOB	FUNCTION	PROCEDURE
BY	GENERATOR	PUBLIC
CASE	GEN_ID	REAL
CAST	GROUP	RIGHT
CHAR	HAVING	ROWS
COALESCE	HOUR	SELECT

COLLATE	IF	SET
COLUMN	IN	SIZE
COMMIT	INACTIVE	SOME
CONTINUE	INDEX	SQL
COUNT	INNER	SUM
CREATE	INSERT	TABLE
CURRENT	INTEGER	THEN
CURRENT_DATE	INTO	TRIGGER
CURRENT_ROLE	IS	TRUNCATE
CURRENT_TIME	JOIN	TYPE
CURRENT_TIMESTAMP	KEY	UNION
CURRENT_TRANSACTION	LAST	UPDATE
CURRENT_USER	LEFT	UPDATING
CURSOR	LENGTH	USE
DATABASE	LIKE	USER
DATE	LONG	USING
DAY	MAX	VALUE
DEC	MERGE	VALUES
DECIMAL	MIN	VIEW
DECLARE	MINUTE	WHEN
DEFAULT	MONTH	WHERE
DELETE	NAMES	WHILE
DESC	NO	WITH
DESCENDING	NOT	YEAR
DISTINCT	NULL	YEARDAY
DO	NULLIF	

Dúvidas Frequentes

Espaço dedicado para a inclusão de dúvidas comuns durante o desenvolvimento em SQL. Caso sua dúvida não esteja aqui, talvez seja interessante adicioná-la.

Quando utilizar parâmetros, QuotedStr ou aspas duplas?

Parâmetros:

Utilizar principalmente para as condições da cláusula

```
SELECT *
  FROM Tabela
 WHERE Campo1 = :PParametro1
   AND Campo2 = :PParametro2
   AND Campo3 = :PParametro3
```

QuotedStr:

Utilizar quando for preciso concatenar uma variável string na consulta

```
SELECT Campo1, Campo2, Campo3
  CASE
    WHEN
('      Campo4 = QuotedStr(LVariavel)');
    THEN
      Campo4
    ELSE
      Campo5
  END ApelidoDoCampo
  FROM Tabela
 WHERE Campo1 IS NOT NULL
```

Aspas duplas:

Utilizar quando uma string fixa for inserida na cláusula

```
SELECT Campo1, Campo2, Campo3
  FROM Tabela
 WHERE Campo1 = "NFE"
```

A utilização da palavra reservada "AS" é obrigatória para apelidar campos ou tabelas?

Qual a diferença entre a utilização de um UNION e UNION ALL?

Qual JOIN devo utilizar na cláusula?

Boas práticas

Parte decisiva em manter um código limpo e de fácil manutenção, as boas práticas são essenciais para trabalhar em equipe em qualquer projeto. Neste capítulo, se encontram sugestões e práticas adotadas para a manutenção do código e uso das ferramentas utilizadas durante todo o processo de desenvolvimento.

Versionamento

Mensagens de Versionamento

Ao realizar qualquer alteração que seja necessário realizar um commit, é importantíssimo adicionar uma mensagem que possa informar do que se trata aquela alteração. Essa mensagem deve ser breve e descritiva, explicando em poucas palavras o que foi alterado, de forma que, caso outro desenvolvedor necessite realizar uma busca pelo log de alterações, as mensagens possam servir de "filtro" para identificar o que é relevante para a sua consulta.

> Deve-se evitar ao máximo realizar commits sem mensagem, principalmente em versões e no trunk.

Exemplos

Modelo de Mensagem

```
Caso XXXXX: Descrição do caso de acordo com o Mantis/Trello.

- Mensagem descritiva e em poucas palavras do que foi alterado

(Opcional) Desenvolvedor
```

Criação de branch

> Ao criar um branch, deve-se informar o número do caso e a sua descrição de acordo com o Mantis/Trello.

```
Caso XXXXX: Descrição do caso de acordo com o Mantis/Trello.

- Criação do branch

(Opcional) Desenvolvedor
```

Alterações

>

> Ao realizar o commit de alterações, além do cabeçalho indicando o número do caso e a descrição, deve-se adicionar por tópico as alterações feitas naquele commit.

```
Caso XXXXX: Descrição do caso de acordo com o Mantis/Trello.

- Alterações na lógica da rotina CalcularSaldo.
- Correções no padrão de código aplicado.
- Criação da classe TGerador

(Opcional) Desenvolvedor
```

Atualização de Branch

> Ao realizar a atualização de um branch(Merge com Trunk) , deve-se adicionar a "tag" [Atualização do Branch] no início do cabeçalho, para que ao consultar o log de alterações, o desenvolvedor identifique prontamente quais revisões são referentes à atualizações do branch e quais não são.

```
[Atualização do Branch] Caso XXXXX: Descrição do caso de acordo com o Mantis/Trello.

(Opcional) Desenvolvedor
```

Documentação Interna

Comentários

Segundo Robert C. Martin em Código Limpo: Habilidades Práticas do Agile Software,

> “O uso adequado de comentários é compensar nosso fracasso em nos expressar no código.”

Dessa maneira, o **uso de comentários no código deve ser evitado**, visto que, conforme o código passar por manutenção, a tendência é que o comentário fique desatualizado e não condizente com o trecho de código à que se refere.

Comentários que podem ser adicionados:

Comentários descritivos:

```
// INSS
if (DM_GERACAO.EventosSendoGerados.IndexOf('e301') <> -1) then
begin
Código Referente ao INSS
end;
```

> Comentários que complementam o entendimento de uma rotina, mas que não tentam explicar o que está sendo feito no trecho.

Comentários que devem ser evitados/removidos:

Comentários no cabeçalho:

```
{******************************************************************
* Módulo : ArqTeste □□□□□□□□□□*
* Finalidade : Realizar a exportação de Notas Fiscais □□□*
* de Serviços para Prefeitura de São Paulo □□□□□□*
* Data : 21/02/2000 □□□□□□□□□□*
* Programador : Fulano da Silva Santos □□□□□□□*
******************************************************************}
```

> Comentários utilizados como cabeçalho que descrevem o que a unit faz, a data de implementação e o programador responsável pela implementação apesar de serem descritivos, muitas vezes não são atualizados conforme o código passa por mudanças. Essas informações podem e devem ser encontradas nas mensagens de versionamento.

Comentários referentes à processos passados:

```
// MIGRACAO
// Caso a classe uClass.NumDocs seja utilizada em alguma unit
// a diretiva referente ao arquivo uClass.NumDocs.inc deverá ser acrescentada.
// Ex: Ver units FormPri e Backup, antes do nome da unit;
// No arquivo uClass.NumDocs.inc deverá ser adicionada a diretiva referente ao
// sistema ao qual a classe NumDocs foi adicionada.
```

> Comentários como esses não acrescentam informações no entendimento do código e acabam criando uma poluição visual na unit, de modo que, com o passar do tempo a tendência é que esse tipo de comentário seja automaticamente ignorado pelos desenvolvedores.

Código comentado:

```
// if ((QMANUT.FieldByName('DiasDireito').AsFloat = 0) or
//□   (SameText(Trim(QMANUT.FieldByName('DiasDireito').AsString), ''))) and
//     (MessageDialog.Show('Não foi informado os dias de Direito. Deseja Continuar?',
//□□  mtConfirmation, [mbYes, mbNo], 0) = mrNo) then
// begin
// □ QMANUT.FieldByName('DiasDireito').FocusControl;
// □ Abort;
// end;
```

Códigos comentados geram confusões no código, além de poluir desnecessariamente a unit, dessa maneira códigos não devem ser comentados e sim removidos. Caso necessário, pode-se visitar o histórico de revisões para acompanhar as alterações do trecho.

Código Limpo

Definição de Clean Code

A apresentação de um código claro e organizado não consiste apenas na convenção nomes, constantes, classes, variáveis, espaçamento etc.. Um código limpo (*clean code*) deve ser:

- Simples: fácil entendimento;
- Eficiente: realizar tudo o que foi proposto;
- Único: não realizar algo que outro trecho de código já faz;
- Direto: não dar voltas para chegar no resultado;
- Feito com atenção: o código deve ser sempre feito com preocupação e revisto depois de pronto;

Bad Smells

Em contra partida, também existem as *Bad Smells*, que como sua tradução já sugere, é algo com cheiro ruim, e representa o código com práticas que não devem ser utilizadas. Essas práticas se devem aos códigos que fogem das características de um código limpo. Abaixo são demonstrados alguns exemplos de *bad smells*.

Encadeamento de If's

```
if (Q_Manut.FieldByName('DiasDireito').AsFloat = 0) then
begin
  if (AnsiSameText(Trim(Q_Manut.FieldByName('DiasDireito').AsString), '')) then
  begin
    if (MessageDialog.Show('Não foi informado os dias de Direito. Deseja' +
          'Continuar?', mtConfirmation, [mbYes, mbNo], 0) = mrNo) then
    begin
      Q_Manut.FieldByName('DiasDireito').FocusControl;
      Abort;
    end;
  end;
end;
```

> Esse tipo de encadeamento pode ser facilmente substituído pela palavra reservada "and".

```
if (Q_Manut.FieldByName('DiasDireito').AsFloat = 0) and
   (AnsiSameText(Trim(Q_Manut.FieldByName('DiasDireito').AsString), ''))) and
   (MessageDialog.Show('Não foi informado os dias de Direito. Deseja' +
      'Continuar?', mtConfirmation, [mbYes, mbNo], 0) = mrNo) then
begin
  Q_Manut.FieldByName('DiasDireito').FocusControl;
  Abort;
end;
```

Excesso de if-else

```
if (Q_Manut.FieldByName('DiasDireito').AsFloat = 0) then
  LTipo := 1
else if (Q_Manut.FieldByName('DiasDireito').AsFloat = 1) then
  LTipo := 2
else if (Q_Manut.FieldByName('DiasDireito').AsFloat = 2) then
  LTipo := 3
else if (Q_Manut.FieldByName('DiasDireito').AsFloat = 3) then
  LTipo := 5
else
  LTipo := 9;
```

> O trecho acima pode ter sua estrutura facilitada com a utilização de um "case".

```
case Q_Manut.FieldByName('DiasDireito').AsFloat of
  0: LTipo := 1;
  1: LTipo := 2;
  2: LTipo := 3;
  3: LTipo := 5;
  else
    LTipo := 9;
end;
```

Atribuição Indireta

> Situações com atribuições indiretas aparecem constantemente no código

```
if (Q_Manut.FieldByName('DiasDireito').AsFloat = 0) then
  LDeveIncrementar := False
else
  LDeveIncrementar := True;
```

```
if (RG_TipoCliente.ItemIndex = 0) then
  LTipo := 0
else (RG_TipoCliente.ItemIndex = 1) then
  LTipo := 1;
```

> Basta uma pequena análise por parte do desenvolvedor para perceber que podem ser simplificadas

```
 LDeveIncrementar := (Q_Manut.FieldByName('DiasDireito').AsFloat <> 0);
```

```
LTipo := RG_TipoCliente.ItemIndex;
```