

Delphi

Borland



Copyright © 2005 - Marcos Antonio Moreira.

Todos os nomes dos produtos citados são marcas registradas da Borland International, Inc.  
Outros produtos citados são marcas registradas no respectivo cabeçalho

As várias Marcas Registradas que aparecem no decorrer deste livro. Mais do que simplesmente listar esses nomes e informar quem possui seus direitos de exploração, ou ainda imprimir o logotipo das mesmas, o autor declara estar utilizando tais nomes apenas para fins editoriais, em benefício exclusivo do dono da marca registrada, sem intenção de infringir as regras de sua utilização.

## INDICE

### Capitulo 1 – Introdução

Introdução à Linguagem de Programação	05
O que é processamento de dados	06
O que é Sistemas Operacionais	06
Como é executado um programa de computador	06

### Capitulo 2 – Ambiente Delphi

O ambiente de desenvolvimento ( IDE )	09
Trabalhando com Form	14
Estrutura de uma Unit	16
Arquivos usados em um projetos	18

### Capitulo 3 – Componentes Básicos

Definição	20
Componentes Visuais	20
Componentes Não Visuais	20
Vantagens do Uso de Componentes	21

### Capitulo 4 – A Linguagem Delphi Language

Símbolos Especiais	23
Palavras reservadas	24
Números	25
Constantes	25
Identificadores	26
Strings de caracteres	26
Comentários	27
Estruturas de controle	27

### Capitulo 5 – Modularização

Procedimento	34
Função	35

### Capitulo 6 – Orientação a Objetos

Conceito	39
Classe	40
Herança	40
Polimorfismo	40
Embutimento	40

# 1

## Introdução

## Introdução à Linguagem de Programação

---

Uma linguagem de programação é um método padronizado para expressar instruções para um Computador. É um conjunto de regras sintáticas e semânticas usadas para definir um programa de computador. Uma linguagem permite que um programador especifique precisamente sobre quais dados um computador vai atuar, como estes dados serão armazenados ou transmitidos e quais ações devem ser tomadas sob várias circunstâncias. O conjunto de palavras (tokens), compostos de acordo com essas regras, constituem o código fonte de um software. Esse código fonte é depois traduzido para código de máquina, que é executado pelo processador.

Código fonte é o conjunto de palavras escritas de forma ordenada, contendo instruções em uma das linguagens de programação existentes no mercado, de maneira lógica. Após compilado, transforma-se em software, ou seja, programas executáveis. Este conjunto de palavras, que formam linhas de comandos, deverão estar dentro da padronização da linguagem escolhida, obedecendo critérios de execução. Atualmente com a diversificação de linguagens, o código pode ser escrito de forma totalmente modular, podendo um mesmo conjunto de códigos ser compartilhado por diversos programas, e até mesmo linguagens.

Software, logicial ou programa de computador é uma sequência de instruções a serem seguidas e/ou executadas, na manipulação, redirecionamento ou modificação de um dado/informação ou acontecimento.

Todo computador possui um conjunto de instruções que seu processador é capaz de executar. Essas instruções são representadas por números([opcodes](#)).Um programa em código de máquina consiste de uma sequência de números que significam uma sequência de instruções a serem executadas. Programar diretamente em código de máquina costuma ser exaustivamente difícil, pois requer o conhecimento dos opcodes de cada instrução.Por esse motivo, foi criada uma linguagem de programação chamada linguagem de montagem ([Assembly Language](#)), composta de códigos mnemônicos que, do ponto de vista técnico, é tão próxima do processador quanto o código de máquina, mas é humanamente mais fácil de se compreender uma vez que seus códigos são geralmente acrônimos do inglês. Por exemplo ´mov´ de mover, ´rep´ de repetição e assim por diante.

## Processamento de Dados

---

Consiste em extrair informação de dados. Os dados são os elementos básicos pertencentes a um conjunto determinado de informações.

Por exemplo um documento de identificação pode conter vários dados de uma pessoa como nome, sexo, data de nascimento, etc. Outros exemplos de dados são a temperatura de uma cidade, ou a área de um território. Ainda que estes pareçam, por vezes, isolados, podem sempre englobar-se em conjuntos (as temperaturas das cidades de uma província ou país, ou as áreas de um conjunto de territórios) ou séries (as temperaturas de uma cidade ao longo do tempo).

## Sistemas Operacionais

---

Sistema Operacional ou Sistema Operativo é um conjunto de ferramentas necessárias para que um computador possa ser utilizado de forma adequada. Consiste na camada intermediária entre o aplicativo e o hardware da máquina.

Este conjunto é constituído por um Kernel, ou núcleo, e um conjunto de softwares básicos, que executam operações simples, mas que juntos fazem uma grande diferença.

Antes que existissem sistemas desse tipo, todo software desenvolvido deveria saber se comunicar com os dispositivos do computador de que precisasse.

## Como é executado um Programa

---

É composto por uma sequência lógica de instruções, que é interpretada e executada por um processador. Um programa pode ser executado por qualquer dispositivo capaz de interpretar e executar as instruções de que é formado. Esses dispositivos possuem um ou mais processadores, que são circuitos complexos e criados para esse fim. Quando um software está escrito usando instruções que podem ser executadas diretamente por um processador dizemos que está escrito em linguagem de máquina. A execução de um software também pode ser intermediada por um programa interpretador, responsável por interpretar e executar cada uma de suas instruções. Uma categoria especial e notável de interpretadores são as máquinas virtuais, como a JVM (Máquina Virtual Java), que simulam um computador inteiro, real ou imaginado.

## **Borland®**

---

O dispositivo mais conhecido que dispõe de um processador é o computador. Existem outras máquinas programáveis, como telefone celular, máquinas de automação industrial, calculadora, etc

# 2

Ambiente  
Delphi ( IDE )

## Ambiente de Desenvolvimento ( IDE )

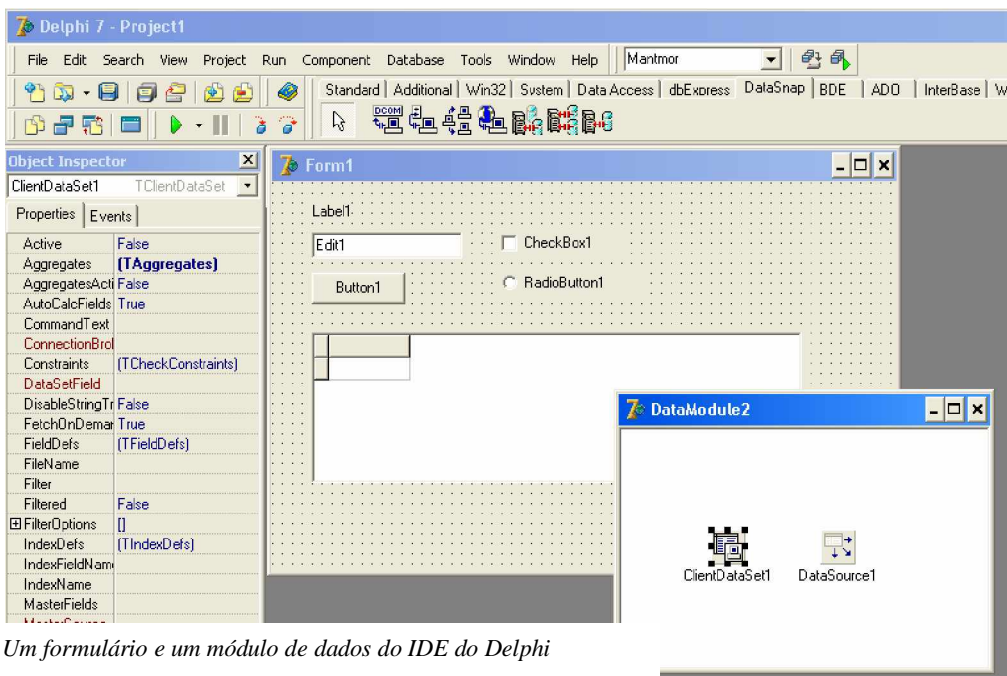
---

Em uma ferramenta de programação visual como o Delphi, o papel do IDE ( Integrated Development Environment – Ambiente integrado de desenvolvimento ) é, às vezes, ainda mais importante do que a linguagem de programação.

O Delphi 7 fornece alguns recursos novos e interessantes que aproveitam o rico IDE do Delphi 6. Se você é um programador iniciante, não se preocupe.

O uso do IDE do Delphi é bastante intuitivo.

Ao trabalhar com um ambiente de desenvolvimento visual, seu tempo é gasto em duas partes diferentes do aplicativo: nos projetista visuais e no editor de código. Os projetista visuais permitem que você trabalhe com os componentes no nível visual (quando você incluir um botão em um formulário) ou no nível não visual (quando você inclui um componente DataSet em um módulo de dados).



*Um formulário e um módulo de dados do IDE do Delphi*

O editor é o local na qual você escreve o código. O mais óbvio de escrever código em um ambiente visual é responder aos eventos, iniciando com aqueles ligados às operações executadas pelos usuários do programa, tal como clicarem um botão ou selecionar um item em uma caixa de lista.

À medida que o programador conhece melhor o Delphi, ele passa a escrever principalmente código de manipulação de eventos e, em seguida, passam a escrever suas próprias classes e componentes, e quase sempre acabam passando a maior parte de seu tempo no editor.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, DBGrids, StdCtrls;

type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Label1: TLabel;
    Button1: TButton;
    CheckBox1: TCheckBox;
    RadioButton1: TRadioButton;
    DBGrid1: TDBGrid;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
begin

end;
```

Editor de códigos

## Configuração da Área de Trabalho

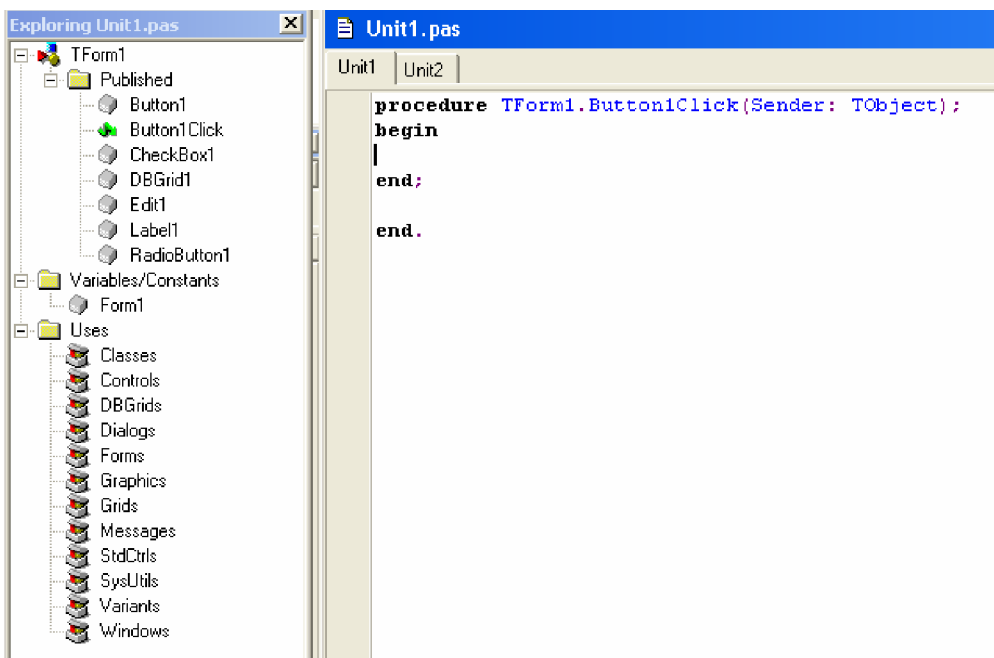
---

A Configuração da Área de Trabalho, pode personalizar o IDE do Delphi de diversas maneiras – em geral, abrindo muitas janelas reorganizando-as entre outras. Por isso, o Delphi permite salvar determinadas organização das janelas do IDE (chamadas de área de trabalho ou desktop)

## Code Explorer

---

O Code Explorer, que geralmente fica acoplado ao lado do editor, lista todos os tipos, variáveis e rotinas definidas em uma unidade, além das outras unidades que apareçam em instruções uses. Você pode usar o Code Explorer para navegar no editor. Se você der um clique duplo em uma das entradas no Code Explorer, o editor pulará para a declaração correspondente.



## Code Explorer

## Completamento d Classe

---

Completamento de Classe (Code Complete ) – O editor do Delphi também pode ajudar gerando parte do código-fonte para você, completando aquilo que você já tem escrito. Esse recurso é chamado de completamento de classe, e você pode ativá-lo pressionando a combinação de teclas Ctrl+Shift+C

```
    DBGrid1: TDBGrid;
private
    { Private declarations }
public
    Procedure Hello (Mensagem: String);
    { Public declarations }
end;

var
    Form1: TForm1;

implementation

{$R *.dfm}
{ TForm1 }

procedure TForm1.Hello(Mensagem: String);
begin
|
end;

end.
```

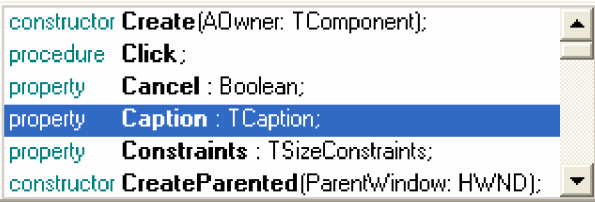
Completamento de Classe

## Completamento de Código

---

Completamento de Código permite escolher a propriedade ou método de um objeto simplesmente procurando-o em uma lista ou digitando suas letras iniciais. Para chamar essa lista, você apenas digita o nome de um objeto, como Button1, insere o ponto e espera. Para forçar a exibição da lista, pressione Ctrl+Barra de espaço

```
procedure TForm1.Button1Click(Sender: TObject);  
begin  
  Button1.  
end;  
end.
```



Completamento de Código

## Form Designer

---

Outra janela do Delphi a qual você vai interagir freqüentemente é o Form Designer, uma ferramenta visual para colocação de componentes.

Os formulários (objeto TForm) são os pontos centrais para o desenvolvimento Delphi. Você se utilizará deles para desenhar sua comunicação com o usuário, colocando e organizando outros objetos. Estes objetos são arrastados da Component Palette, mostrada na janela localizada acima.

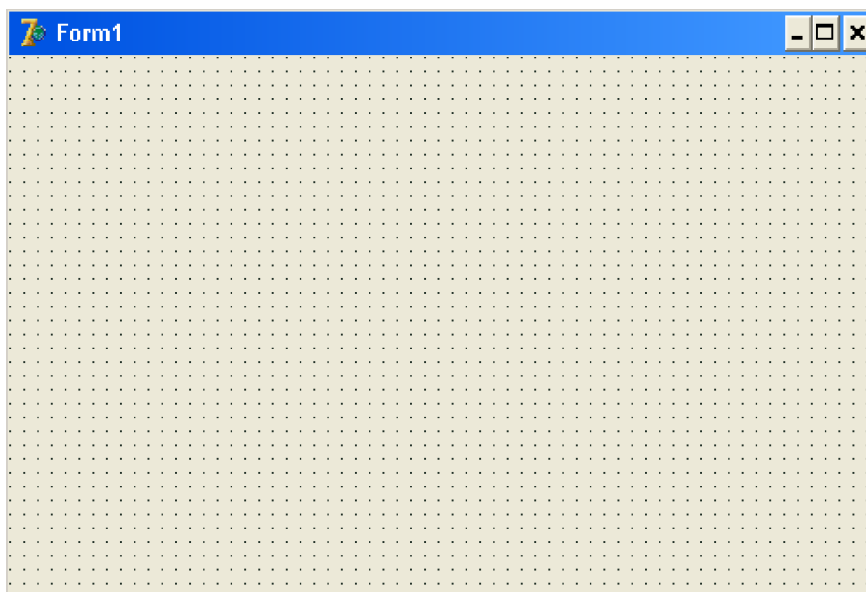
Você pode imaginar que o formulário é um objeto que contém outros objetos. Sua aplicação ficará localizada em um formulário principal e este interagirá com outros formulários criados. É possível aumentar, mover ou ocupar completamente a tela do monitor, ou até mesmo ultrapassá-la.

Um formulário básico inclui os seguintes componentes:

- Û Controles de menu;
- Û Botões de maximização e minimização;
- Û Barra de título; e
- Û Bordas redimensionáveis.

O código gerado, na área conhecida como Code Editor, fica exatamente atrás do objeto formulário, clique na barra de notas, em Unit1.

Ao terminar de projetar um formulário, você pode usar o comando Lock Controls do menu edit para impedir a mudança acidental da posição de um componente.




## Code Editor

---

O editor de códigos providência total acesso ao código gerado pelo projeto, incluindo alguns dos mais poderosos recursos para a edição. Pode ser selecionado tipos de cores para os elementos do programa (como por exemplo comentários, palavras reservadas, códigos assembler, ...) para tanto a partir do menu principal entre em **T**ools | **O**ptions..., localize a página Colors.

Ao ser aberto um novo projeto, o Delphi gera automaticamente na página do Code Editor uma Unit com o arquivo código (.PAS). Para ver o código de uma Unit em particular, simplesmente Click na tabulação de página.

O Code Editor mostrará sempre o nome do arquivo corrente ativo na tabulação de página.

É possível alternar entre o objeto Form e a Code Editor através do pressionamento da tecla F12, do botão  (Toggle Form/Unit) da SpeedBar, ou ainda através das opções do menu **V**iew | **T**oggle Form/Unit. (curiosidade: o acesso rápido através da tecla Alt + Letra sublinhada para esta opção está marcado sobre a letra G)

## Object Inspector

---

Providência a conexão entre a interface visual e o código. É Composto por duas páginas Properties (propriedades) e Events (Eventos) que mostrará as propriedades e eventos do objeto selecionado.

Disponibiliza um fácil caminho para a personalização dos objetos. Você usará a página de Propriedades para personalizar os objetos colocados no formulário (inclusive o próprio formulário), e a página de Eventos para gerenciar a navegação entre certas partes do código do programa.

O seletor de objetos (Object Selector - localizado em um objeto do tipo ComboBox no topo do Object Inspector) mostra o nome e o tipo de todos os componentes do formulário corrente (inclusive o próprio). Você pode usar o seletor de objetos para localizar facilmente qualquer objeto no formulário.

## Estrutura de Uma Unit

---

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs;

type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

end.
```

### Seção Unit

Declara o nome da unit.

### Seção Uses

Contém as units acessadas por este arquivo.

### Seção Interface

Nesta seção estão as declarações de constantes, tipos de variáveis, funções e procedures gerais da Unit/Form. As declarações desta seção são visíveis por qualquer Unit. Esta seção é formada pelo seguinte código:

INTERFACE - Palavra que inicia a seção;

USES - Cláusula que inicia uma lista de outras unidades compiladas (units) em que se baseia:

SysUtils = utilitários do sistema (strings, data/hora, gerar arquivos)

WinProcs = acesso a GDI, USER e KERNEL do Windows

Wintypes= tipos de dados e valores constantes

Messages=constantes com os números das mensagens do Windows e tipos de dados das Mensagens

Classes=elementos de baixo nível do sistema de componentes

Graphics=elementos gráficos

Controls=elementos de nível médio do sistema de componentes

Forms=componentes de forma e componentes invisíveis de aplicativos

Dialogs=componentes de diálogo comuns

## Seção Type

Declara os tipos definidos pelo usuário. Subseções: Private, declarações privadas da Unit. Public declarações públicas da Unit.

## Seção Var

Declara as variáveis privadas utilizadas.

## Seção Implementation

Contém os corpos das funções e procedures declaradas nas seções Interface e Type. Nesta seção também estão definidos todos os procedimentos dos componentes que estão incluídos no Form. As declarações desta seção são visíveis apenas por ela mesma. Esta seção é formada pelo seguinte código:

{R\*.DFM} - Diretiva compiladora que inclui toda a interface, propriedades da forma e componentes do arquivo \*.DFM

{S+} - Diretiva compiladora que ativa verificação de pilha.

## Seção uses adicional

Serve para declarar Units que ativam esta.

## Inicialization

Nesta seção, que é opcional, pode ser definido um código para proceder as tarefas de inicialização da Unit quando o programa começa. Ela consiste na palavra reservada inicialization seguida por uma ou mais declarações para serem executadas em ordem.

## Arquivos Produzidos pelo Sistema

Extensão	Tipo de arquivo e descrição	Momento da Criação	Requerido para a compilação
.CFG	Arquivo de configuração com opções de projeto.	Desenvolvimento	Requerido apenas se as opções especiais do compilador foram configuradas
.RES, .RC	Arquivos de recursos: o arquivo binário associado ao projeto e que, em geral contém seu ícone	Caixa de diálogo Development Options	Sim. O arquivo RES principalmente de um aplicativo é reconstruído pelo Delphi.
.DOF	Delphi Option File (arquivo de opções o Delphi): um arquivo de texto com as configurações atuais das opções de projeto	Desenvolvimento	Requerido apenas se as opções especiais do compilador foram configuradas.
.DPR	Arquivo Delphi Projet: (Projeto Delphi: na verdade, este arquivo contém código-fonte Delphi)	Desenvolvimento	Sim
.DFM	Delphi Form File (arquivo de formulário do Delphi): arquivo binário com a descrição das propriedades de um formulário ou de um módulo de dados	Desenvolvimento	Sim. Cada formulário é armazenado em um arquivo PAS e em um arquivo DFM.
.PAS	Arquivo fonte de uma unidade Delphi Language (Pascal), ou uma unidade relacionada com um formulário ou unidade independente	Desenvolvimento	Sim
.DSK, .DLL, .EXE, BPL			

# 3

## Componentes Básicos do Delphi



## Vantagens do Uso de Componentes

---

Com o advento da programação visual o uso de componentes tornou-se um grande atrativo. Fica muito mais prático e rápido você adicionar um botão em seu formulário e alterar as suas propriedades de forma a, no mesmo instante, observar os efeitos desta alteração.

Além disso o uso de componentes torna o desenvolvimento de aplicações muito mais rápido. Uma vez que você já possui um componente testado e confiável, basta adiciona-lo em sua aplicação e ajustar as suas propriedades que ele se encarrega de fazer o restante. Ele já sabe o que deve fazer. Se, por exemplo, um usuário alterar o sistema de cores de sua máquina, você não precisa se preocupar em criar rotinas que alterem as cores de sua aplicação, pois os componentes que usam o sistema de cor fazem a alteração necessária.

Outro aspecto importante é que, com o uso de componentes, a manutenção de um sistema se torna mais eficiente. Imagine o trabalho que daria se depois que você acabou de criar uma aplicação, descobre um pequeno problema nos botões, e tem que fazer uma alteração em todos os locais onde aparece algum botão!

# 4

## A Linguagem Delphi

A linguagem Delphi Language (Pascal) é uma linguagem Orientada a Objetos não pura mas híbrida por possuir características de programação não só visual mas também escrita, para os programadores que já conhecem técnicas de estruturas de programação, com o C, Basic, Pascal ou xBASE entre outras linguagens a Delphi Language providência uma migração de forma natural oferecendo um produto de maior complexibilidade. Delphi Language força a você executar passos lógicos isto torna mais fácil o desenvolvimento no ambiente Windows® de aplicações livres ou que utilizam banco de dados do tipo Client/Server, Webservice, Multi Camada e Web, trabalha com o uso de ponteiros para a alocação de memória e todo o poder de um código totalmente compilável. Além disso possibilita a criação e reutilização (vantagem de re-uso tão sonhado com a Orientação a Objetos) de objetos e bibliotecas dinâmicas (Dynamic Link Libraries - DLL).

[AL1] Comentário:

Delphi Language contém todo o conceito da orientação a objetos incluindo encapsulamento, herança e polimorfismo. Algumas extensões foram incluídas para facilitar o uso tais como conceitos de propriedades, particulares e públicas, e tipos de informações em modo run-time, manuseamento de exceções, e referências de classes. O resultado de toda esta junção faz com que Delphi Language consiga suportar as facilidades de um baixo nível de programação, tais como:

- ü Controle e acesso das subclasses do Windows® (API);
- ü Passar por cima das mensagens de loop do Windows®;
- ü Mensagens semelhantes as do Windows®;
- ü Código puro da linguagem Assembler.

Como deu para perceber a base de toda a programação Delphi é a linguagem Object Pascal, então neste capítulo trataremos exclusivamente deste tipo de programação.

## Símbolos Especiais

---

A Delphi Language aceita os seguintes caracteres ASCII:

- ü Letras - do Alfabeto Inglês: A até Z e a até z.
- ü Dígitos - Decimal: 0 até 9 e Hexadecimal: 0 até 9 e A até F (ou a até f)
- ü Brancos - Espaço (ASCII 32) e todos os caracteres de controle ASCII (ASCII 0 até ASCII 31), incluindo final de linha e Enter (ASCII 13).
- ü Especiais - Caracteres: + - \* / = < > [ ] . , ( ) : ; ^ @ { } \$ #
- ü Símbolos - Caracteres: <= >= := .. (\* \*) (. .) //

O colchete esquerdo ( [ ) e equivalente ao ( . e o colchete direito ( ] ) e equivalente a ). A chave esquerda ( { ) e equivalente ao ( \* e a chave direita ( } ) e equivalente a \*).

## Palavras Reservadas

---

A Delphi Language se utiliza das seguintes palavras reservadas, não podendo as mesmas serem utilizadas ou redefinidas:

And	Exports	Library	Set
Array	File	Mod	Shl
As	Finally	Nil	Shr
Asm	For	Not	String
Begin	Function	Object	Then
Case	Goto	Of	To
Class	If	On	Try
Const	Implementation	Or	Type
Constructor	In	Packed	Unit
Destructor	Inherited	Procedure	Until
Div	Initialization	Program	Uses
Do	Inline	Property	Var
Downto	Interface	Raise	While
Else	Is	Record	With
End	Label	Repeat	Xor
Except			

Uma outra lista a seguir, apresenta as diretivas que são utilizadas em contextos de identificação de objetos:

Absolute	Export	Name	Published
Abstract	External	Near	Read
Assembler	Far	Nodefault	Resident
At	Forward	Override	Stored
Cdecl	Index	Private	Virtual
Default	Interrupt	Protected	Write
Dynamic	Message	Public	

## Números

---

É possível definir variáveis e constantes de tipos de Inteiro ou Real através de qualquer decimal ordinário ( 0 a 9 ), mas a Delphi Language também aceita a notação Hexadecimal utilizados com o prefixo dollar ( \$ ) ou a notação científica ( E ).

## Constantes

---

Uma constante é um identificador com valor(es) fixo(s). Um bloco de declarações constante possui a seguinte expressão:

[Declaração Constante] [Identificador] (=) [constante] (:)

A lista abaixo apresenta um conjunto de funções que podem ser utilizadas para a declaração das constantes:

Ab	Length	Ord	SizeOf
Chr	Lo	Pred	Succ
Hi	Low	Ptr	Swap
High	Odd	Round	Trunc

Alguns exemplos para a definição de Constantes:

```
const Min = 0;  
Max = 100;  
Centro = (Max - Min) div 2;  
Beta = Chr(225);  
NumLetras = Ord('Z') - Ord('A') + 1;  
MensOla = 'Instrução inválida';  
MensErro = ' Erro: ' + MensOla + ' . ' ;  
PosErr = 80 - Length(MensErro) div 2;  
Ln10 = 2.302585092994045684;  
Ln10R = 1 / Ln10;  
DigNumericos = ['0'..'9'];  
LetrasAlpha = ['A'..'Z', 'a'..'z'];  
AlphaNum = LetrasAlpha + DigNumericos;
```

## Expressões

---

As expressões em Delphi Language (como em qualquer linguagem) é formada por operadores e operandos; os operadores são divididos em quatro categorias básicas:

Únicos	@, Not
Multiplicativos	>, /, div, mod, and, shl, shr, as
Adicionais	+, -, or, xor
Relacionais	=, <, >, <=, >=, <>, in, is

As expressões obedecem as regras básicas de lógica para a precedência da execução das operações.

## Identificadores

---

Identificadores podem ser constantes, tipos, variáveis, procedures, funções, unidades, programas e campos de registros.

Não existe limite de caracteres para o nome de um identificador mas apenas os 63 primeiros caracteres são significantes (não podendo ser idêntico ao nome das palavras reservadas). O nome de um identificador deve ser iniciado por Letras ou o carácter underscore ( \_ ). O resto é formado por Letras, Dígitos, carácter underscore (ASCII \$5F). Não é permitido a utilização de espaços para a formação do nome.

Exemplo de identificadores válidos: Form1, SysUtils.StrLen, Label1.Caption

## Strings

---

Strings são conjunto de caracteres entre aspas simples.

Exemplos:

`'isto é uma string'`.

`'123456'`.

Etc.

## Comentários

---

Comentários são textos que introduzimos no meio do programa fonte com a intenção de torná-lo mais claro. É uma boa prática em programação inserir comentários no meio dos nossos programas. No Turbo Pascal, tudo que estiver entre os símbolos

(\* e \*) ou { e } ser considerado como comentário.

## Concatenação

---

Esta operação, representada pelo sinal de adição, ou seja, +. Os operandos devem ser do tipo string ou char.

Exemplo:

'Isto é uma ' + 'String' = 'Isto é uma String'.

## Atribuição

---

variavel := valor;

## Estrutura de Controle - Condicional

---

O comando condicional if pode ser composto de uma ou mais condições de processamento, por exemplo:

- if (A > B) then  
  B := B + 1; // ou INC(B);
- if (A > B) then  
  B := B + 1  
  else  
  A := A - 1; // ou DEC(A);
- if (A > B) then  
  begin  
  B := B + 1;  
  X := B + A;  
  end

```
else
  begin
    A := A - 1;
    Y := Y + B;
  end;
```

No último exemplo para representar um bloco de comandos em caso verdadeiro ou falso, utiliza-se dos delimitadores *begin* e *end*.

O comando *if-then-else* é considerado como único, portanto, não há ponto e vírgula (;) antes da palavra reservada *else*.

## Estrutura de Controle - Repetição

---

```
for <variavel> := <valor_inicial> to <valor_final> do
  <comando1>;
```

ou

```
for <variavel> := <valor_inicial> downto <valor_final> do
  <comando1>;
```

## Estrutura de Controle - Repeat

---

O comando *repeat..until* é uma opção para estruturas de repetição. A grande diferença com o comando *while* é o fato do comando *repeat* ser executado pelo menos uma vez.

```
repeat
  X := X + 1;
  INC(Z,3); //equivale a Z := Z + 3;
  DEC(AUX,2);
until X >= 200;
```

```
repeat
  <comandos>;
until <condição>;
```

Nas estruturas while e for, se houver mais de um comando a ser processado, eles devem ser colocados entre BEGIN e END.

## Estrutura de Controle - While

---

O comando while..do também permite a construção de estruturas de repetição, com diferença de não executar o laço no início do teste lógico.

```
while X <= 200 do
begin
  X := X + 1;
  INC(Z,3);
  DEC(AUX,2);
end;
```

```
while <condição> do
begin
  <comando1>;
  <comando2>;
  <comandon>;
end;
```

## Estrutura de Controle - For

---

O comando for..do estabelece uma estrutura de repetição considerando um controle inicial e final. Pode ser construído de maneira crescente ou decrescente.

```
for i:=0 to 500 do
  Label1.Caption := IntToStr(i);
```

```
for i:=500 downto 100 do
begin
  Label1.Caption := IntToStr(i);
  Edit1.Caption := IntToStr(i);
end;
```

```
for <variavel> := <valor_inicial> to <valor_final> do
begin
  <comando1>;
  <comando2>;
  <comandon>;
end;
```

## Estrutura de Controle - Escolha

---

O comando `case..of` oferece uma alternativa para comandos `if-then-else` com um 'grande' número de testes. Por exemplo:

```
case Key of
  'A'..'Z':      Label1.Caption := 'Letras';
  '0'..'9':      Label1.Caption := 'Números';
  '+', '-', '*', '/': Label1.Caption := 'Operador'
else
  Label1.Caption := 'Caracter especial';
end; //fim do case
```

## Estrutura de Controle - With

---

`with... do...;`

Delimita um determinado bloco de declarações para um identificador específico evitando a declaração deste identificador. A sintaxe do comando é: `WITH {nome do identificador} DO {comandos};`. Ex:

```
begin
  { ... comandos iniciais ... }
  with form1 do
    begin
      Caption := 'Teste';           ò Equivalente a Form1.Caption
      BorderStyle := bsSizable;    ò Equivalente a Form1.BorderStyle
    end;
  end;
end;
```

O comando `with..do` é usado para abreviar a referência a campos de registro, ou a métodos, e propriedades de um objeto.

```
begin
  Form1.Caption := 'Senac';
  Form1.Color := CIBlue;
  Form1.Top := 95;
end;

//Equivalente à:
with Form1 do
  begin
    Caption := 'Senac';
    Color := CIBlue;
    Top := 95;
  end;
```

## Vetores - Array

---

Define um conjunto de variáveis ou constantes de um mesmo tipo. A sintaxe do comando é: `array [{quantidade de ocorrências}] of {Tipo};`. Os arrays são controlados por três funções:

### var

**array** [ lim\_inf .. lim\_sup ] of <tipo>;

– Exemplo:

```
var
  Alunos : array [1..100] of string;
  Alunos[1] := 'Valeria';
```

## Declarações

---

Declarações descrevem ações de um algoritmo a serem executadas.

`begin... end;`

Prende um conjunto de declarações em um bloco de comandos determinado. A sintaxe do comando é: `BEGIN {comandos} END;`. Ex:

```
begin
  { ... comandos iniciais ... }
  begin
    { ... bloco 1 ... }
  end;
  begin
    { ... bloco 2 ... }
  end;
  { ... comandos finais ... }
end;
```

## Registro

---

```
var
  nome_registro : record
  campo : tipo;
end;
```

ou

type

```
nome_registro = record  
campo : tipo;  
end;
```

Exemplo

```
var  
CadFunc : record  
    nome, endereco : string;  
    cpf : string;  
    sexo : char;  
    TemDependentes : boolean;  
end;
```

# 5

## Modularização

Para construirmos grandes programas, necessitamos fazer uso da técnica de modularização. Esta técnica faz com que dividamos um grande programa em pequenos trechos de código, onde cada qual tem uma função bem definida. Assim, além da facilidade em lidar com trechos menores, ainda podemos fazer uso da reutilização de código, já que estes trechos devem ser bem independentes.

Assim, definimos módulo como um grupo de comandos, constituindo um trecho de algoritmo, com uma função bem definida e o mais independente possível em relação ao resto do algoritmo.

A maneira mais intuitiva de trabalharmos com a modularização de problemas é definir-se um módulo principal de controle e módulos específicos para as funções do algoritmo. Módulos de um programa devem ter um tamanho limitado, já que módulos muito grandes são difíceis de serem compreendidos.

Os módulos são implementados através de procedimentos ou funções.

Sintaxe de definição de um procedimento

**Procedimento** Nome\_Procedimento [ (parâmetros) ];

**declare**

< variáveis locais >

**início**

comando 1;

comando 2;

comando n;

**fim**

Os parâmetros podem ser passados por valor ou por referência. Um parâmetro passado por valor, não pode ser alterado pelo procedimento. Os parâmetros por referência são identificados usando-se a palavra VAR antes de sua declaração

Os valores passados como parâmetros na chamada de um procedimento, devem corresponder sequencialmente à ordem declarada.

A função é semelhante ao procedimento, todavia sua diferença consiste no fato de que um procedimento não retorna valor quando é chamado para execução, enquanto que a função retorna um valor.

Definição de uma função:

```
Função Nome_Função [ (parâmetros) ] : valor_retorno;  
declare  
    < variáveis locais >  
início  
    comando 1;  
    comando 2;  
    comando n;  
fim
```

Chamada da função:

Nome\_Função (<lista de parâmetros>);

Ou

Variável ← Nome\_Função (<lista de parâmetros>);

Procedimentos e funções são blocos de código (rotinas) em Object Pascal que podem ou não receber parâmetros (valores) para processamento. Uma vez definida a rotina pode-se ativa-la de diversas partes do programa através de seu nome.

A grande diferença entre as formas de definição destas rotinas (se procedimentos ou funções) está no fato de que:

- ü Procedimento – NÃO retorna valor.
- ü Função – Retorna valor.

## Declaração e Ativação de Procedimento

---

Podemos declarar um procedimento da seguinte maneira maneira:  
Dentro da cláusula `private` ou `public`, defina a declaração do procedimento:

```
procedure Soma(X, Y: String);
```

Com o cursor posicionado na mesma linha, pressione: `CTRL+SHIFT+C` e perceba que o próprio Delphi realiza a construção do procedimento dentro da cláusula `implementation`. Esse recurso é chamado `Class Completion`. Nossa tarefa é apenas definir o código a ser realizado pelo procedimento.

```
procedure TForm1.Soma(X, Y: String);  
begin  
    Label1.Caption := FloatToStr(StrToFloat(X)+StrToFloat(Y));  
end;
```

Supondo a existência de dois componentes `Edit`, um componente `Button` e um componente `Label`, este código pode ser ativado da seguinte forma:

```
procedure TForm1.Button1Click(Sender: TObject);  
begin  
    Soma(Edit1.Text, Edit2.Text);  
end;
```

## Declaração e Ativação de Funções

---

A construção de funções tem o mesmo raciocínio diferindo na característica de retorno da função.

Podemos declarar um procedimento da seguinte maneira maneira:  
Dentro da cláusula `private` ou `public`, defina a declaração da função:

```
function Subtrai(X, Y: String): String;
```

Observe que agora, depois dos parâmetros há um tipo de definição de retorno da função (`String`).

Pode-se utilizar a mesma dica de construção do procedimento, na linha da declaração tecla `CTRL+SHIFT+C` (`Class Completion`) e perceba que o próprio Delphi realiza a construção da função dentro da cláusula `implementation`.

Nossa tarefa é apenas definir o código a ser realizado pela função.

```
function TForm1.Subtrai(X, Y: String): String;  
begin  
    result := FloatToStr(StrToFloat(X)-StrToFloat(Y));  
end;
```



A palavra reservada `result` é o recurso usado pela Object Pascal para estabelecer o retorno da rotina. Não se deve declarar esta variável, ela é declarada no momento da utilização da função.

Supondo a existência de dois componentes `Edit`, 'um' componente `Button` e um componente `Label`, esta função pode ser ativada da seguinte forma:

```
function TForm1.Button2Click(Sender: TObject);  
begin  
    Label1.Caption := Subtrai(Edit1.Text, Edit2.Text);  
end;
```

Neste caso, o `Label` recebe o `result` de `subtrai`, ou seja, a subtração dos dados passados nos parâmetros.

# 6

## Orientação a Objetos - Eventos

## Conceito de Programação Orientado a Objeto

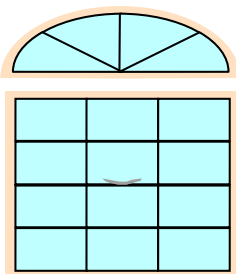
---

Para compreendermos melhor a novo ambiente de desenvolvimento da Borland o Delphi é necessário que você, aprenda e, tenha em mente os conceitos de POO (Programação Orientada a Objetos), não confunda os conceitos com POE (Programação Orientada a Eventos) muito difundido com o Access 2.0<sup>®</sup> (um ambiente baseado em Objetos), mas ao longo deste capítulo você vai notar as sensíveis diferenças que existem entre esses dois conceitos.

A POO e a POE são facilmente confundidas, mas lembre-se a POO contém a POE mas a POE não contém a POO, um objeto pode existir mesmo que não exista nenhum evento associado a ele, mas um evento não pode existir se não houver um objeto a ele associado. Outra característica que pode causar confusão são ambientes Orientados a Objetos e ambientes Baseados em Objetos. Em ambiente Orientado a Objetos consegue-se criar e manipular objetos enquanto que o Baseado em Objetos não é possível a criação de objetos apenas a sua manipulação.

A POO é um conceito desenvolvido para facilitar o uso de códigos de desenvolvimento em interfaces gráficas. Sendo a Borland, uma das primeiras a entrar neste novo conceito, possui suas principais linguagens de programação (tais como Object Pascal e C++), totalmente voltadas para este tipo de programação. A POO atraiu muitos adeptos principalmente pelo pouco uso de código que o projeto (diferente de sistema) carrega no programa fonte, ao contrário das linguagens mais antigas como o Clipper'87<sup>®</sup> muito utilizado no final da década de 80 e início da década de 90. O resultado desta "limpeza" no código resulta que a manutenção do projeto torna-se muito mais simples.

Antes de começarmos a falar realmente de linguagem orientada a objetos e necessário que você possua os conceitos básicos da orientação a objetos, são eles:



**Objeto** - Um objeto é uma instancia de uma classe, ou uma variável do tipo de dado definido pela classe. Os objetos são entidades reais. Uma janela por exemplo, é um objeto de uma casa, de um carro ou de um software com interface gráfica para o usuário.

**Atributos** - São as características do objeto, como cor e tamanho, a janela, por exemplo, tem atributos como o modelo, tamanho, abertura simples ou dupla, entre outros.

**Encapsulação** - é um mecanismo interno do objeto "escondido" do usuário. Uma pessoa pode abrir uma janela girando a tranca sem precisar saber o que há dentro dela.

**Ação** - é a operação efetuada pelo objeto. Todas as janelas, por exemplo, controlam a iluminação e temperatura ambiente, dependendo do seu design.

## Herança

---

Um objeto novo nem sempre é criado do zero. Ele pode “herdar” atributos e ações de outros já existentes. Um basculante herda atributos das janelas e das persianas.

## Polimorfismo

---

É a capacidade de objetos diferentes reagirem segundo a sua função a uma ordem padrão. O comando “abre”, por exemplo, faz um objeto entrar em ação, seja ele uma janela, uma porta ou uma tampa de garrafa.

## Classe

---

Uma classe é um tipo de dado definido pelo usuário, a qual tem um estado (sua representação ou dados internos) e algumas operações (seu comportamento ou seu métodos).

## Encapsulamento

---

Uma classe pode ter qualquer quantidade de dados e qualquer número de métodos. Entretanto, para uma boa estratégia orientada a objetos, os dados devem ser ocultos ou encapsulados dentro da classe que os está usando.

## Delphi Language

---

Para melhor compreendermos o que seja uma classe, devemos começar definindo dois elementos fundamentais na codificação de um software. São as estruturas de dados, ou seja, áreas de memórias utilizadas para armazenar os dados usados por nossos programas, e os trechos de código que manipulam estas estruturas. Essencialmente, uma classe nada mais é do que uma estrutura que conterà dados e códigos que manipulará estes dados.

O Delphi se baseia nos conceitos de POO e, em particular, na definição de novos tipos de classes. O uso de POO é parcialmente implantado pelo ambiente de desenvolvimento visual, porque para cada formulário definindo em tempo de projeto, o Delphi define automaticamente uma nova classe. Além disso, cada componente colocado visualmente em um formulário é um objeto de um tipo de classe disponível ou incluído na biblioteca do sistema.

Como acontece em outras linguagens de POO mais modernas (entre elas o Java e o C#), no Delphi uma variável tipo classe não fornece o armazenamento

Do objeto, mas é apenas um ponteiro ou uma referência para o objeto na memória. Antes de usar o objeto, você deve alocar memória para ele criando uma instância nova ou atribuindo uma instancia existente para a variável.

Var

```
Objeto1, objeto2: TMinhaClasse;
```

Begin

```
// Atribui um objeto recém-criado
```

```
Objeto1 : TMinhaClasse.Create;
```

```
// atribui a um objeto existente
```

```
Objeto2 := ObjetoExistente;
```

A chamada de create invoca um construtor-padrão disponível para cada classe, a menos que a classe o redefina. Para declara um novo tipo de dados de classe no Delphi, com alguns campos de dados locais e alguns métodos, você usa a sintaxe a seguir:

Type

```
TData = class
```

```
    Mês, Dia, Ano: Integer;
```

```
    Procedure SetValue (m,d,a: Integer);
```

```
    Function LeapYear: Boolean;
```

```
End;
```

Um método é definido com a palavra-chave function ou procedure, dependendo de ele ter ou não valor de retorno. Na definição de classe, os métodos se podem ser declarados. Em seguida, eles devem ser definidos na parte de implementação (implementation) da mesma unidade (Unit).

```
Function Tdate.LeanYear: Boolean;
```

Begin

```
// chama isleapYear em SysUtils.pass
```

```
    Result := IsLeapYear(Year);
```

End;

A seguir é apresentado como você pode usar um objeto da classe definida anteriormente:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TData = class
    Mes, Dia, Ano: Integer;
    procedure SetValue(m, d, y: Integer);
    function LeapYear: Boolean;
  end;
  TForm1 = class(TForm)
    Button1: TButton;
    Edit1: TEdit;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    m, d, a : Word;
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

uses DateUtils;

{$R *.dfm}

{ TData }

function TData.LeapYear: Boolean;
begin
  Result := IsLeapYear(ano);
end;

procedure TData.SetValue(m, d, y: Integer);
begin
  Mes := m;
  Dia := d;
  Ano := y;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  Aday: TData;
begin
  // Cria o Objeto;
  Aday := TData.Create;
  Try
    Aday.SetValue(1,1,2000);
    If Aday.LeapYear then
      ShowMessage('Ano Bissextto:' + IntToStr(Aday.Ano));
  Finally
    Aday.Free;
  End;
end;

end.
```

Observe que `Aday.LeapYear` é uma expressão semelhante a `Aday.Year`, embora a primeira seja uma chamada de função e a segunda, um acesso direto

a dados. Opcionalmente, você pode incluir parênteses após a chamada de uma função sem parâmetros.

## Encapsulamento

---

O conceito de encapsulamento é indicado com frequência pela idéia de "caixa-preta" Você não conhece o funcionamento inteiro: você só sabe como fazer a interface com a "Caixa-Preta" ou como usa-la, independentemente da sua estrutura interna. A parte referente a 'como usar', chamada de interface da classe, permite que outras partes de um programa acessem e usem os objetos daquela classe. Entretanto, quando você usa objetos, a maior parte de seu código fica oculta. Entretanto, no Delphi há um nível extr de ocultamento, realizado por meio das propriedades, como veremos mais adiante.

## Private, Protected e Public

---

Para o encapsulamento com base em classe, a linguagem Delphi tem três especificadores de acesso: Private, Protected e Public. Um quarto, Published para tempo de projeto ou melhor fica em propriedades de um componente e pode ser comparado ao public.

Private => A diretiva private denota campos e métodos de uma classe, os quais não são acessíveis fora da unidade(unit) que declara a classe.

Protected => É usada para indicar métodos e campos com visibilidade limitada. Apenas a classe atual e suas subclasses podem acessar elementos protected. Mais precisamente, apenas a classe, as subclasses e o código da mesma unidade da classe podem acessar os membros protegidos.

Public => Denota campos e métodos que são livremente acessíveis por meio de qualquer parte de um programa, assim como na unidade na qual eles são definidos.

Em geral, os campos de uma classe devem ser privados e os métodos normalmente são públicos.

Como exemplo, considere esta nova versão da classe TData:

unit Dates;

interface

uses

    SysUtils;

type

    TDate = class

    private

        fDate: TDateTime;

        procedure SetDay(const Value: Integer);

        procedure SetMonth(const Value: Integer);

        procedure SetYear(const Value: Integer);

        function GetDay: Integer;

        function GetMonth: Integer;

        function GetYear: Integer;

    public

        procedure SetValue (y, m, d: Integer); overload;

        procedure SetValue (NewDate: TDateTime); overload;

        function LeapYear: Boolean;

        function GetText: string;

        procedure Increase;

        property Year: Integer read GetYear write SetYear;

        property Month: Integer read GetMonth write SetMonth;

        property Day: Integer read GetDay write SetDay;

    end;

implementation

uses

    DateUtils;

procedure TDate.SetValue (y, m, d: Integer);

begin

    fDate := EncodeDate (y, m, d);

end;

function TDate.GetText: string;

begin

    Result := DateToStr (fDate);

end;

procedure TDate.Increase;

begin

    fDate := fDate + 1;

end;

```
function TDate.LeapYear: Boolean;
begin
    // from DateUtils
    Result := IsInLeapYear(fDate);
end;

procedure TDate.SetValue(NewDate: TDateTime);
begin
    fDate := NewDate;
end;

procedure TDate.SetDay(const Value: Integer);
begin
    fDate := RecodeDay (fDate, Value);
end;

procedure TDate.SetMonth(const Value: Integer);
begin
    fDate := RecodeMonth (fDate, Value);
end;

procedure TDate.SetYear(const Value: Integer);
begin
    fDate := RecodeYear (fDate, Value);
end;

function TDate.GetDay: Integer;
begin
    Result := DayOf (fDate);
end;

function TDate.GetMonth: Integer;
begin
    Result := MonthOf (fDate);
end;

function TDate.GetYear: Integer;
begin
    Result := YearOf (fDate);
end;

end.
```

---

```
unit DateF;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, Dates, StdCtrls;

type
  TDateForm = class(TForm)
    BtnIncrease: TButton;
    EditYear: TEdit;
    EditMonth: TEdit;
    EditDay: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    BtnRead: TButton;
    BtnWrite: TButton;
    procedure BtnIncreaseClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure BtnReadClick(Sender: TObject);
    procedure BtnWriteClick(Sender: TObject);
  private
    TheDay: TDate;
  public
    { Public declarations }
  end;

var
  DateForm: TDateForm;

implementation

{$R *.DFM}

procedure TDateForm.FormCreate(Sender: TObject);
begin
  TheDay := TDate.Create;
  TheDay.SetValue(2003, 12, 25);
end;

procedure TDateForm.BtnIncreaseClick(Sender: TObject);
begin
  TheDay.Increase;
end;

procedure TDateForm.FormDestroy(Sender: TObject);
begin
  TheDay.Free;
end;

procedure TDateForm.BtnReadClick(Sender: TObject);
begin
  EditYear.Text := IntToStr(TheDay.Year);
  EditMonth.Text := IntToStr(TheDay.Month);
  EditDay.Text := IntToStr(TheDay.Day);
end;
```

```
procedure TDateForm.BtnWriteClick(Sender: TObject);
begin
  TheDay.SetValue (StrToInt (EditYear.Text),
  StrToInt (EditMonth.Text), StrToInt (EditDay.Text));
end;

end.
```

## Eventos

---

Os programas feitos em Delphi são orientados a eventos. Eventos são ações normalmente geradas pelo usuário. Ex.: Clicar o mouse pressionar uma tecla, mover o mouse etc. Os eventos podem ser também gerados pelo windows.

Existem eventos associados ao formulário e cada componente inserido neste. Exemplos:

- Ao formulário está ligado on create, que ocorre quando mostramos o formulário na tela.
- Ao componente botão está ligado o evento on click, que ocorre quando damos um click com o mouse sobre o botão.
- 

Eventos comuns ao formulário e aos componentes.

Alguns eventos ligados tanto ao formulário quanto aos componentes estão listados a seguir.

- ü OnClick: ocorre quando o usuário clica o objeto.
- ü OnDblClick: ocorre quando o usuário dá um duplo clique.
- ü OnKeyDown: ocorre quando o usuário pressiona uma tecla enquanto o objeto tem foco.
- ü OnKeyUp: ocorre quando o usuário solta uma tecla enquanto o objeto tem o foco.
- ü OnKeyPress: ocorre quando usuário dá um clique numa tecla ANSI.
- ü OnMouseDown: ocorre quando o usuário pressiona o botão do mouse.
- ü OnMouseUp: ocorre quando o usuário solta o botão do mouse.
- ü OnMouseMove: ocorre quando o usuário move o ponteiro do mouse.

### Rotinas que Respondem a Eventos

Cada evento gera uma procedure, aonde você deve inserir as linhas de código que envolvem este evento. Por exemplo, o evento OnClick, que é gerado ao clicarmos em um botão chamado BTNSair, cria a procedure:

Procedure TForm1.BTNSairClick(Sender: TObject);

onde TForm1 é o objeto TForm que contém o botão BTNSair, e Sender é um objeto TObject que representa o componente que deu origem ao evento.

Se você quiser inserir uma rotina que trate um determinado evento de um componente, faça o seguinte:

- ü clique sobre o componente;
- ü no Object Inspector, seleciona a página Events;
- ü dê um duplo clique sobre o evento para o qual quer inserir o código;
- ü entre no editor de código e escreva as linhas de código.

Exemplo:

```
Procedure TForm1.BTNSairClick(Sender: TObject);
begin
  Form1.Close;
end;
```

Obs.: Escreva seu código entre o begin e o end, se por acaso você quiser retirar o evento e o componente, retire primeiro os eventos do componente removendo somente o código que você colocou e depois o componente; os resto dos procedimentos o DELPHI tira para você.

## Propriedades

---

Como vimos, eventos podem estar associados a modificações em propriedade de componente e formulário, ou seja, você pode modificar propriedades de formulários e componentes durante a execução do sistema. Para isto você deverá usar a sintaxe:

<componente>.<propriedade>;

Por exemplo, para modificar a propriedade text de uma caixa de edição Edit1 para "Bom Dia" faça:

```
Edit1.Text := 'Bom Dia';
```

Se a propriedade do componente tiver sub propriedades, para acessá-lá, utilize a seguinte sintaxe:

<componente>.<propriedade>.<subpropriedade>

Por exemplo, para modificar a sub propriedade Name referente a propriedade fonte, de uma caixa de edição Edit1, para 'Script', faça:

```
Edit1.Font.name := 'Script';
```

Obs.: Verifique o tipo da propriedade para antes de mandar o valor, consultando no Objetc Inspector.