Universidade Federal da Paraíba
Centro de Ciências Sociais Aplicadas
Departamento de Ciência da Informação
Programa de Pós-Graduação em Ciência da Informação
Ação para cidadania e acesso livre à informação

Rede de projetos do
Laboratório de Tecnologias Intelectuais – Lt***i***

Projeto Competências em informação *on line*
- Tutoriais em Tecnologias Intelectuais –

TUTORIAL DELPHI

Autor:

Edberto Ferneda

João Pessoa, Maio de 2011

Universidade Federal da Paraíba
Centro de Ciências Sociais Aplicadas
Departamento de Ciência da Informação
Programa de Pós-Graduação em Ciência da Informação
Ação para cidadania e acesso livre à informação

PRÓ-REITORIA DE EXTENSÃO E ASSUNTOS COMUNITÁRIOS - PRAC

PROJETO DE EXTENSÃO**: Competências em informação - Tutoriais em Tecnologias Intelectuais para disseminação da informação na web.**

Coordenador/Orientador:
Prof. Dr. Wagner Junqueira de Araújo

Co-Orientadoras:
Prof.ª. Dra. Isa Maria Freire
Prof.ª. Ms. Alba Lígia de Almeida Silva

Equipe:
Andreonni Medeiros Di Lorenzo
Cristiana da Silva Dantas
Daniela Silva Borges de Oliveira
Enelucia Santos da Silva
Maria Janienne Alves Pereira de Medeiros

Contato para dúvidas e suporte on-line:
Email: projetodeextensaotutoriais@gmail.com
Skype: contatotutoriaisnaweb

Universidade Federal da Paraíba
Centro de Ciências Sociais Aplicadas
Departamento de Ciência da Informação
Programa de Pós-Graduação em Ciência da Informação
Ação para cidadania e acesso livre à informação

Programação Orientada a Objetos

– POO –

João Pessoa, Maio de 2011

1

Orientação a Objetos

O termo "orientado a objetos" tem sido tão usado que o seu sentido concreto se perdeu. Tem caracterizado desde modelos de administração a métodos de auto-ajuda. Muitos produtos de softwares que alegam ser orientados a objetos não o são. Às vezes possuem alguns recursos baseados em objetos, outras vezes nem isso. Toda essa confusão de conceitos fez com que ninguém hoje soubesse mais o que realmente vem a ser orientado a objetos.

A linguagem utilizada pelo Delphi (Object Pascal) é uma linguagem orientada a objetos. Para melhor entender o Delphi é importante compreender os conceitos da programação orientada a objetos (POO).

Apesar de ser um assunto muito badalado, o conceito de POO não é novo. Tem quase trinta anos de idade. Foi introduzido pela primeira vez na linguagem Simula-67. Desde então surgiram várias outras linguagens com esses conceitos como SmallTalk Eiffel. C++, a mais popular, na verdade é uma linguagem híbrida. É uma extensão do C com suporte a classes e objetos, mas permite que se desenvolva programas sem classes ou objetos.

Porque Usar Objetos?

As vantagens de se usar objetos como blocos para a construção de aplicações são muitas. Pode-se citar:

- **Simplicidade:** Os objetos escondem a complexidade do código. Pode-se criar uma complexa aplicação gráfica usando botões, janelas, barras de rolagem, etc.. sem conhecer detalhes do código fonte utilizado para criá-los.
- **Reutilização de código:** Um objeto, depois de criado, pode ser reutilizado por outras aplicações, Ter suas funções estendidas e serem usados como blocos fundamentais em sistemas mais complexos.

Elementos do Modelo Orientado a Objetos

Os autores divergem quanto às características que fazem uma linguagem ser orientada a objetos, mas a maioria concorda que o paradigma se baseia em quatro princípios básicos: abstração, encapsulamento, herança e polimorfismo.

Abstração

É o processo de extrair as características essenciais de um objeto real. A abstração é essencial para se ter um modelo fiel da realidade. O conjunto de características resultante da abstração forma um tipo abstrato de dado, com informações sobre estado e comportamento.

A abstração nem sempre produz os mesmos resultados. Depende do contexto onde é utilizado. Ou seja, a abstração de um objeto por uma pessoa pode ser diferente na visão de outra. Por exemplo, para um mecânico, um carro é caracterizado pela marca, modelo, potência, etc. Já um usuário comum o caracteriza pela cor, marca modelo, preço. E finalmente, para o DETRAN, um carro é totalmente caracterizado pela placa, proprietário, marca, modelo, cor e número de chassi.

Encapsulamento

É o processo de combinar tipos de dados e funções relacionadas em um único bloco de organização e só permitir o aceso a eles através de métodos determinados.

Por exemplo, existe um número determinado de coisas que se pode fazer com um toca-fitas. Pode-se avançar, voltar, gravar, tocar, parar, interromper e ejetar a fita. Dentro do toca-fitas, porém, há varias outras funções sendo realizadas como acionar o motor, desligá-lo, acionar o cabeçote de gravação, liberar o cabeçote, e outras operações mais complexas. Essas funções são escondidas dentro do mecanismo do toca fitas e não temos acesso a elas diretamente. Quando apertamos o "play" o motor é ligado e o cabeçote de reprodução acionado, mas não precisamos saber como isso é feito para usar o toca fitas.

Uma das principais vantagens do encapsulamento é esconder a complexidade do código. Outra, é proteger os dados, permitindo o acesso a eles apenas através de métodos. Dessa maneira evita-se que seus dados sejam corrompidos por aplicações externas.

Herança

É o aproveitamento e extensão das características de uma classe já existente.

Existem muitos exemplos de herança na natureza. No reino animal, os mamíferos herdam a característica de terem uma espinha dorsal por serem uma subclasse dos vertebrados. Acrescentam-se a essas, várias outras características como ter sangue quente, amamentar, etc. Os roedores herdam todas as características dos vertebrados e mamíferos e acrescentam outras, e assim por diante.

Em programação, a herança ocorre quando uma classe aproveita a implementação (estrutura de dados e métodos) de uma outra classe para desenvolver uma especialização dela. É formada então uma hierarquia de classes, onde cada nível é uma especialização do nível anterior, que é mais genérico.

Polimorfismo

É a propriedade de se utilizar um mesmo nome para fazer coisas diferentes.

Por exemplo, mandar alguém correr pode produzir resultados diferentes dependendo da situação. Se a pessoa estiver parada e em pé, irá obedecer à ordem simplesmente correndo. Ser a pessoa estiver no volante de um carro, o resultado será o aumento da pressão do pé no acelerador.

Em programação, uma mesma mensagem poderá provocar um resultado diferente, dependendo dos argumentos que foram passados e do objeto que o receberá. Por exemplo: o envio de uma instrução "**`desenhe()`**' para uma subclasse de "**`polígono`**", poderá desenha um triângulo, retângulo, pentágono, etc. dependendo da classe que a receber.

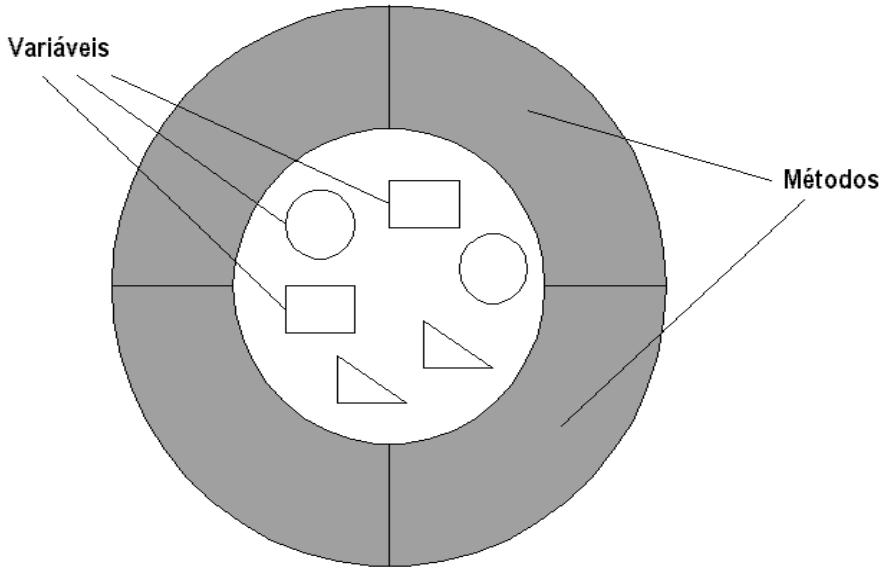
O que é um Objeto?

Objetos são a base da tecnologia orientada a objetos. Consistem de modelos (abstrações) de objetos reais, com seus estados e seus comportamentos.

Qualquer objeto real pode ser abstraído, filtrando-se seus estados e comportamentos.

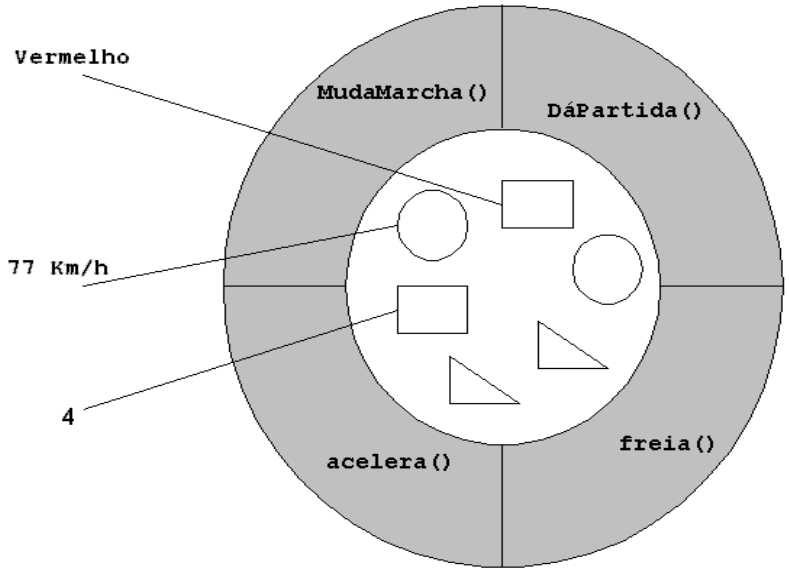
Objeto	Estado	Comportamento
Carro	Velocidade atual, marcha, cor, modelo, marca	troca marchas, acelera, dá partida, freia.
Gato	nome, raça, com fome, com preguiça	mia, dorme, se estica, brinca, caça.
Caneta	cor, nível da tinta	escreve, coloca mais tinta
Toca-fitas	ligado, sentido da fica, tocando, gravando	grava, toca, avança, volta, para, pausa, ejeta, liga.

Em programação, o estado de um objeto é representado por variáveis, e seu comportamento é implementado com métodos (procedimentos). Os métodos são o único meio de mudar o estado das variáveis. Por exemplo, para mudar o estado do toca-fitas para ligado, é necessário utilizar o método **`liga().`**



Tudo o que o objeto de software sabe (seu estado) e tudo o que ele pode fazer (comportamento) é determinado pelas suas variáveis e métodos.

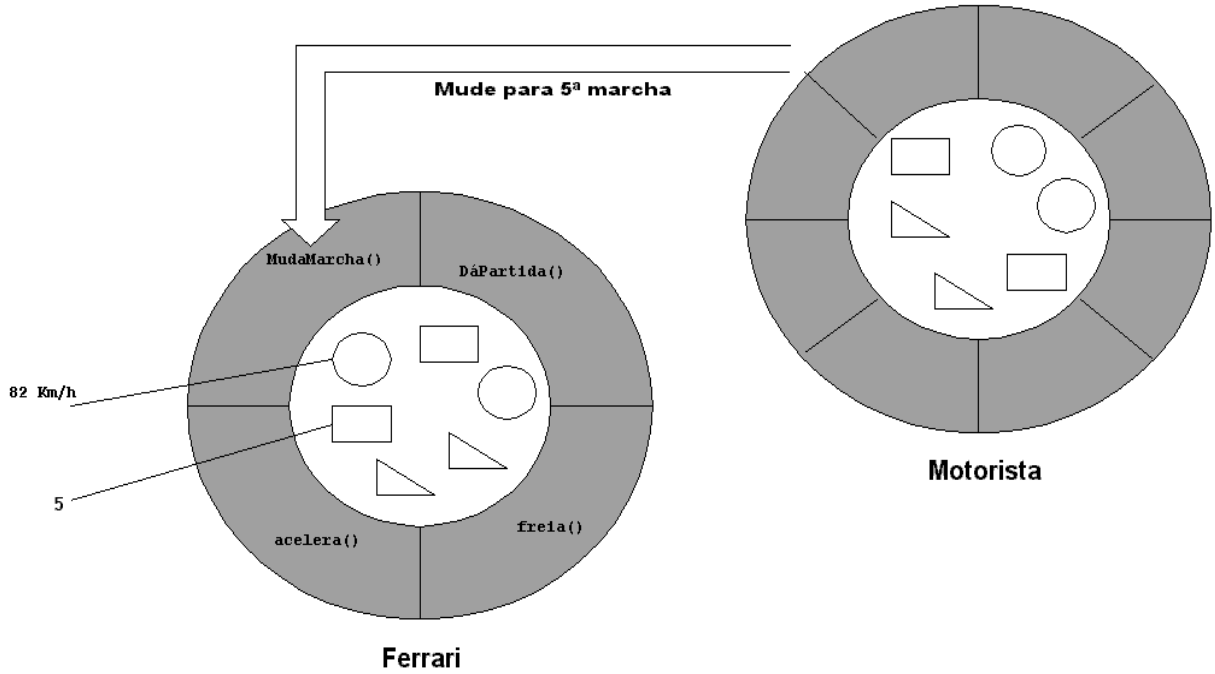
Vamos representar o modelo de um automóvel através de um objeto de software. As variáveis indicam o seu estado atual: **`Velocidade=77 Km/h`**, **`marcha=4`** e **`cor=vermelho`**. **`DáPartida()`**, **`acelera()`**, **`trocaMarcha()`** e **`freia()`** são métodos que podem mudar o estado das variáveis (velocidade e marcha). O objeto que vamos construir se chamará **`ferrari`**.



Um objeto sozinho não tem muita utilidade. Um carro sozinho, sem motorista, na garagem, só serve de enfeite. Para se conseguir alguma funcionalidade dos objetos, é

necessário que haja uma interação com outros objetos. Só assim se pode construir modelos de comportamento mais complexo.

Objetos de software se comunicam entre si através de mensagens. No exemplo do automóvel, o objeto motorista envia uma mensagem para o objeto **`ferrari`**, invocando o método **`trocaMarcha()`**, com o parâmetro **`5`** (quinta-marcha). Com esta informação, o método desejado é executado, provocando a mudança das variáveis do objeto **`ferrari`**.



Para realizar a mudança acima em Delphi (Object Pascal), dentro do código do objeto motorista haveria uma linha do tipo:

```
ferrari.trocaMarcha(5)
```

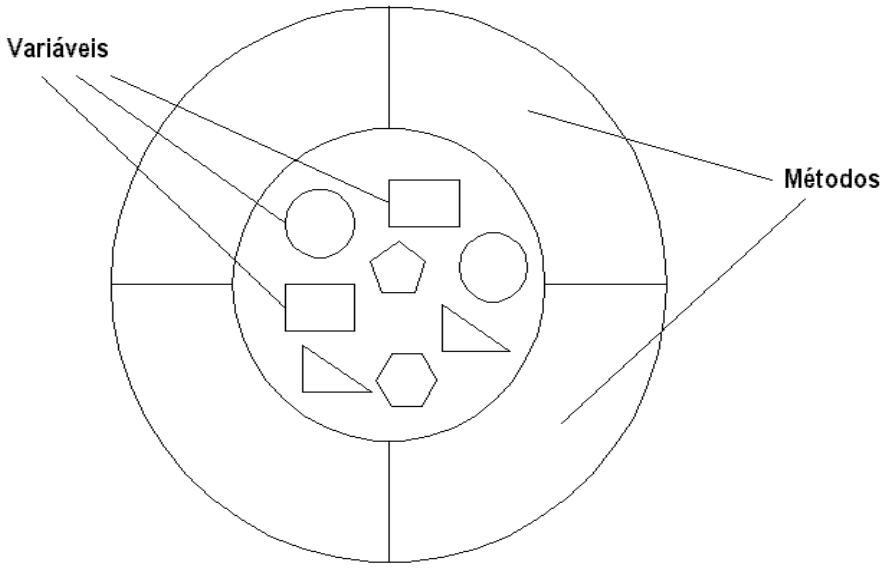
É assim que o Object Pascal representa a invocação do método de um objeto. O primeiro nome (**`ferrari`**) é o objeto. Após o ponto, vem uma variável (atributo/propriedade) ou método.

O que são Classes?

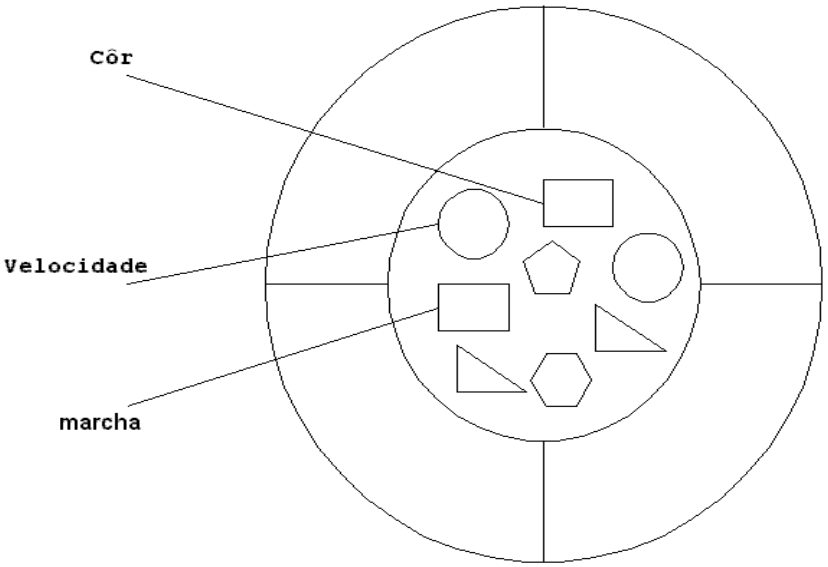
Classes são protótipos utilizados para construir objetos. As características de estado e comportamento do objeto **`ferrari`**, ilustrado nos exemplos anteriores, são

características de qualquer automóvel que sai da fábrica. Essas características são definidas nas classes.

A figura abaixo representa uma classe. Observe a semelhança com o diagrama que utilizamos para representar os objetos. A classe não é um objeto pois não possui um estado, nem um comportamento específico. Ela não é algo que existe e que possua estados.



Para produzir todos os automóveis que saem da fábrica, poderíamos definir uma classe chamada **`Automóvel`**. Esta classe teria a declaração das variáveis que serão utilizadas para armazenar a cor, velocidade e marcha, além da implementação dos métodos que cada automóvel deve ter.



Os objetos são instâncias das classes. Quando criamos um objeto, dizemos que instanciamos uma classe. Cada instância é uma entidade individual e pode então receber valores para as suas variáveis e modificá-las através de métodos.

Com a classe **`Automóvel`**, podemos criar agora vários automóveis com as mesmas características básicas. Por exemplo, se **`ferrari`**, **`fusca`**, **`porsche`** são instâncias de **`Automóvel`**, todos possuem uma velocidade, uma posição de marcha e uma cor, e podem dar partida, acelerar, freiar e trocar marcha. **`Porsche`** pode ser vermelho e estar a 193 Km/h, enquanto **`fusca`** pode ser preto e estar parado.

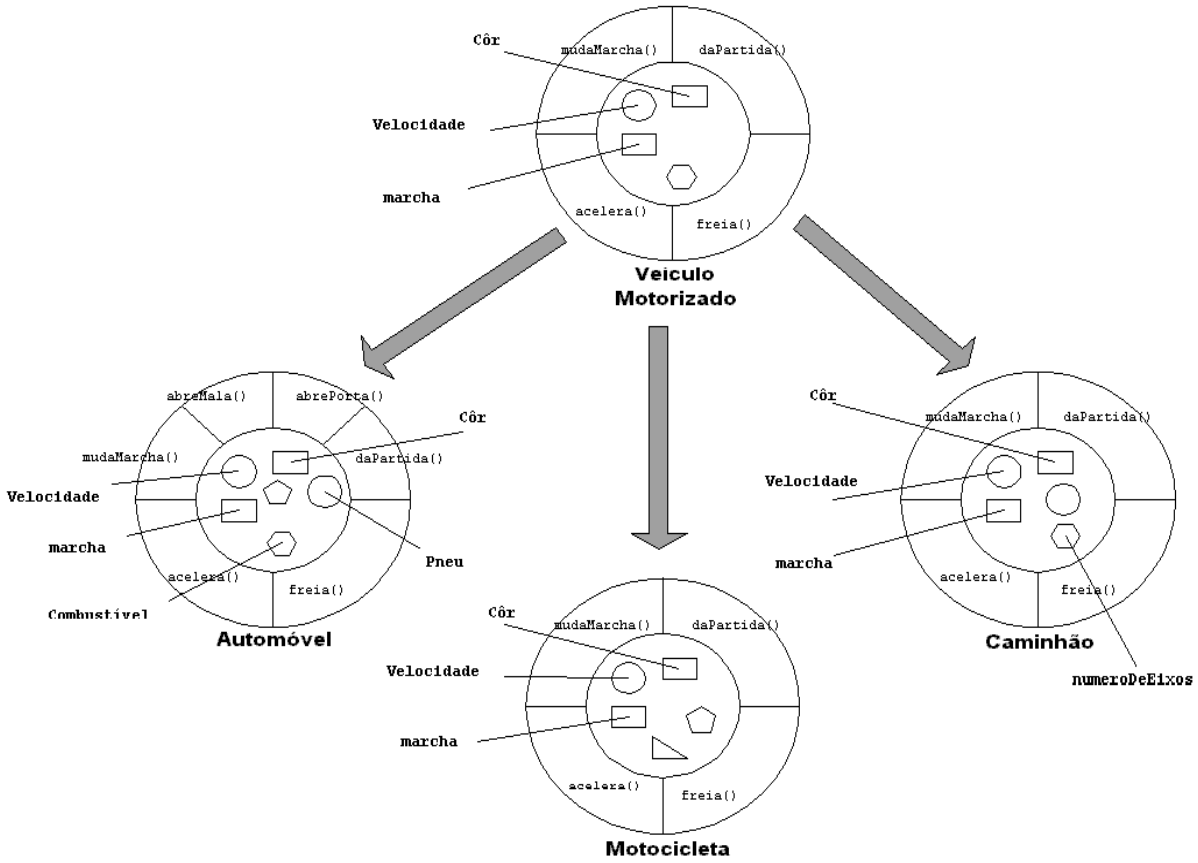
A grande vantagem de se usar classes é a reutilização do código. Toda a estrutura de um objeto é definida na classe, que depois é utilizada quantas vezes for necessário para criar vários objetos.

Herança

Não bastasse a reutilização do código proporcionado pelas classes, na criação de objetos, a POO vai mais longe e define meios de permitir em uma classe, a reutilização do código de outras classes.

As características cor, velocidade, marcha, etc. podem estar definidas em uma classe chamada de Veículo Motorizado, que declara métodos **`trocaMarcha()`**, **`dáPartida()`**, **`acelera()`**, etc. Esses métodos e varáveis podem ser herdadas

pelas classes **Automóvel**, **Motocicleta** e **Caminhão** que são veículos motorizados.



Desta maneira, está formada uma hierarquia de classes. Veículo Motorizado é superclasse de **Automóvel**, **Caminhão** e **Motocicleta**, que por sua vez, são subclasses de **Veículo Motorizado**. O código é escrito uma única vez e usado por várias classes diferentes.

Cada classe herda características de sua superclasse, mas não está limitada à implementação oferecida nela. A subclasse é uma especialização da superclasse e pode definir novas variáveis, novos métodos e até sobrepor métodos definidos na superclasse.

Automóvel, por exemplo, pode acrescentar novos métodos como **abrirPorta()** e **abrirMala()**. **Motocicleta** deve sobrepor o método **acelera()**, **trocaMarcha()**, **dáPartida()** e **freia()** com uma implementação específica para motos. **Caminhão** pode acrescentar novas variáveis

capacidadeDeCarga, númeroDeEixos, etc. para caracterizar melhor os objetos que serão produzidos.

Delphi e a POO

Para se definir uma classe no DELPHI deve-se empregar a seguinte **sintaxe**:

```
type
classe = class(superclasse)
       atributos
       métodos
end;
```

atributos: seguem as regras das declarações de variáveis;

métodos: seguem as regras das declarações de subprogramas (*function* e *procedure*).

No DELPHI:

- A classe mãe é denominada **TObject**. Esta classe deve ser utilizada quando a classe não tiver bem definida sua classe mãe.
- A definição de uma classe deve ser feita na área de **interface** de uma **UNIT**.
- Convenciona-se iniciar o nome de uma classe com a letra **T**.

A definição da classe **TPessoa**, em DELPHI, seria:

```
type
TPessoa = class(TObject)
    Nome : string;
    Endereço : string;
    procedure AtualizarDados (Nome, Endereço : string);
end;
```

No entanto esta definição não está completa, pois deve-se definir o **qualificador de acesso** permitido para cada atributo ou método.

Qualificadores de Acesso

No DELPHI pode-se definir o tipo de acesso que uma classe-objeto terá sobre os atributos ou métodos de outra classe-objeto. Os principais qualificadores de acesso em DELPHI são:

- ***Private***: as declarações feitas nesta área não podem ser acessadas por outras classes-objetos;
- ***Protected***: as declarações feitas nesta área só podem ser acessadas pelas classes filhas da classe-objeto;
- ***Public***: as declarações feitas nesta área podem ser acessadas por qualquer outra classe-objeto.

Sendo assim, uma definição mais completa da **sintaxe** de uma classe, em DELPHI é:

```
classe = class(superclasse)
       private
        {declarações privativas}
       protected
        {declarações protegidas}
       public
        {declarações públicas}
end;
```

Uma nova definição para a classe **TPessoa** poderia ser:

```
classe = pessoa(TObject)
       private
          Nome: String;
          Endereco: String;
       protected
          {declarações protegidas}
       public
          procedure AtualizaDados( Nome, Endereco: String);
end;
```

Definindo-se os atributos Nome e Endereço como *private* eles só podem ser alterados pelo método AtualizarDados, que por sua vez poderá ser acessado por qualquer outra classe-objeto.

No entanto esta definição não está completa. Uma classe-objeto em DELPHI precisa ser criada ou destruída através de **Construtores e Destrutores.**

Construtores e Destrutores

Quando uma nova classe é definida em DELPHI é necessário declarar **construtores**. Um construtor permite **criar** a instância de uma classe, ou seja, um objeto. Convenciona-se chamar um construtor pelo nome de *Create*. Um construtor é denotado pela palavra ***Constructor***.

Ao contrário dos construtores, os **destrutores** são utilizados para **liberar** a instância da classe. Ao destrutor convenciona-se o nome de *Destroy*. Um destrutor é denotado pela palavra ***Destructor***.

Sendo assim, uma nova definição para a classe **TPessoa** poderia ser:

```
type
TPessoa = class(TObject)
    private
      Nome : string;
      Endereco : string;
    public
      procedure AtualizarDados(Nome, Endereco : string);
    protected
      constructor Create;
      destructor Destroy;
end;
```

O *constructor Create* foi colocado na área *protected*. Isto significa que ele só pode ser acessado pelas subclasses da classe TPessoa. Deste modo, a classe TPessoa nunca poderia ser um objeto, pois seu construtor não pode ser acessado por outras classes-objetos.

Poderíamos considerar esta definição como definitiva. No entanto, falta agora a **implementação** dos métodos AtualizarDados e Create.

Implementação de Classes

A implementação das classes deve ser feita na área de **implementation** de uma UNIT, e segue as regras normais da linguagem Turbo Pascal. A seguir é apresentada a implementação completa da classe **TPessoa**.

```
unit Pessoas;
interface
type
  TPessoa = class(TObject)
    protected
      Nome     : string;
      Endereco : string;
      constructor Create;
      destructor Destroy:
    public
      procedure AtualizarDados (Nome, Endereco: string);
end;

implementation
constructor TPessoa.Create;
begin
  inherited Create;
    Nome := '';
    Endereco := '';
end;

destructor TPessoa.Destroy;
begin
  inherited Destroy;
end;

procedure TPessoa.AtualizarDados(Nome, Endereco: string);
begin
  Self.Nome := Nome;
  Self.Endereco := Endereco;
end;
end.
```

Três detalhes importantes devem ser notados na implementação da classe TPessoa: o **constructor Create** e as cláusulas **inherited** e **self**.

Constructor Create

No constructor *Create* podem ser omitidos as inicializações dos atributos. Eles devem aparecer apenas quando deseja-se inicializar um atributo com um determinado valor.

Inherited

A cláusula **inherited**, no *constructor Create*, indica que o método *Create* foi herdado da classe-mãe (no caso, *TObject*).

Self

A cláusula **self**, na *procedure AtualizarDados*, indica que o atributo referenciado é o da classe, e não a variável passada como parâmetro.

2

Object Pascal

A linguagem Object Pascal é proveniente da linguagem Pascal, que durante a década de 80 fez grande sucesso comercial através do Turbo Pascal. A estrutura básica da linguagem, a maioria de seus comandos e sua sintaxe elegante ainda permanecem. Porém, com a evolução para a programação orientada a objetos, a forma de se trabalhar com a linguagem e algumas de suas características foram sendo alteradas. Passou-se do Turbo Pascal 7 para o Turbo Pascal para Windows, que não fez o mesmo sucesso.

Com o Delphi, a linguagem Pascal assume outra vez o lugar de destaque que sempre mereceu. Agora com o nome Object Pascal, muitas de suas características, que muitos pensam ser fruto de atualizações posteriores, na verdade já faziam parte da linguagem padrão, idealizada e concebida pelo seu criador, Niklaus Wirth.

Comentários

Como ponto inicial, você deve saber como fazer comentários em seu programa Pascal. O Delphi suporta três tipos de comentários:

```
{ Comentários utilizando chaves }
(* Comentários usando parente e asterisco *)
// Comentários no estilo C++
```

Variáveis

A linguagem Object Pascal requer que se declare todas as variáveis, antes de se iniciar uma *procedure*, *function* ou *program*. Isto significa que o código escrito em Object Pascal, obedece a uma estrutura um pouco mais rígida do que em outras linguagens:

```
Procedure Foo;
Var
   x, y: integer;
   f: double;
Begin
   X := 1;
   Inc(x);
   y := 2;
// ...
end;
```

A linguagem Object Pascal, diferentemente das linguagem C e C++, não consideram maiúsculas e minúsculas como formas diferentes. Por exemplo, **X** e **x** se referem a uma mesma variável.

Dica

A utilização de letras MAIÚSCULAS e minúsculas pode ser útil para dar clareza aos nomes de variáveis, *procedures*, *functions*, etc. Por exemplo, a seguinte declaração:

Procedure estaprocedurenaofaznada;

é difícil de ler. Ficaria bem melhor da seguinte forma:

Procedure EstaProcedureNaoFazNada;

Note como a Object Pascal permite você agrupar mais de uma variável de mesmo tipo juntas em uma mesma linha:

```
x, y: Integer;
```

Uma característica implementada a partir da versão 2.0 do Delphi, permite se inicializar variáveis **globais** na própria declaração.

```
Var
  I: integer = 10;
  P: Pointer = Nil;
  S: String = 'Alo mamae!';
  D: Double = 3.141579
```

A inicialização prévia de variáveis é permitida apenas para variáveis globais e não aquelas que são locais a uma *procedure* ou *function*.

Constantes

As constantes são definidas após a cláusula **`const`**, similar à linguagem C, como no exemplo abaixo:

```
const float AdecimalNumber = 3.14;
const int i = 10;
const char *ErrorStrng = "Perigo, perigo, perigo!";
```

A diferença entre as constantes entre C e Object Pascal é que em Pascal não é preciso declara o tipo de uma constante. O compilador Delphi aloca automaticamente o espaço apropriado para a constante baseado em seu valor.

```
Const
      AdecimalNumber = 3.14;
      i = 10;
      ErrorString = 'Perigo, perigo, perigo!';
```

Opcionalmente pode-se especificar o tipo de uma constante.

```
Const
      AdecimalNumber: double = 3.14;
      i: Integer = 10;
      ErrorString: string = 'Perigo, perigo, perigo!';
```

A linguagem Object Pascal permite o uso de algumas funções nas declarações de constantes e variáveis. Estas funções incluem **`Ord()`**, **`Chr()`**, **`Trunc()`**, **`Round()`**, **`High()`**, **`Low()`** e **`SizeOf()`**.

```
Type
  A = array[1..2] of integer;
Const
  W: word = Sizeof(Byte);
Var
  I: integer = 8;
  J: SmallInt = ord('a');
  L: LongInt = Trunc(3.14159);
  X: ShortInt = Roud(2.17828);
  B1: byte = High(A);
  B2: byte = Low(A);
  C: Char = Chr(46);
```

Se durante a execução de um programa tentar-se alterar o valor de uma constante, o Delphi apresenta um erro de compilação.

Operadores

Os operadores são símbolos que habilitam a manipulação de todos os tipos de dados. Em Object Pascal os operadores estão definidos basicamente em: *operador de atribuição*, operadores *aritméticos*, *operadores lógicos*, *operadores relacionais*, etc.:

Operador de Atribuição

O operador de atribuição é utilizado para alterar o conteúdo das variáveis. Para atribuir um valor a uma variável utiliza-se o operador **:=**.

```
PI := 3.1415;
Mens := 'Pressione OK para sair'
```

Operador de Comparação

O Object Pascal utiliza o operador **=** para realizar comparações lógicas entre duas expressões ou dois valores.

```
if X = Y then
...
```

Operadores Lógicos

O Pascal utiliza as palavras **and**, **or** e **not** como operadores lógicos.

```
If (condição1) and (condição2) then
      FazIsso
```

```
While (condição1) or (condição2) do
      FazAquilo
```

```
If not (condição) then
      FacaAlgumaCoisa;
```

Operador	Símbolo
Atribuição	`:=`
Comparação	`=`
Diferença	`<>`
Menor	`<`
Maior	`>`
Menor ou igual	`<=`
Maior ou igual	`>=`
e lógico	`and`
ou lógico	`or`
not lógico	`not`

Operadores Aritméticos

De forma geral, os operadores aritméticos são os mesmos existentes em outras linguagens de programação. A diferença é que o Pascal possui diferentes operadores de divisão. O operador **`div`** (divisão inteira) trunca automaticamente o resultado final da divisão.

Operador	Símbolo
Adição	`+`
Subtração	`-`
Multiplicação	`*`
Divisão Real	`/`
Divisão Inteira	`Div`
Resto da Divisão	`Mod`

Atenção

Use o operador de divisão correto para os tipos de expressões que se está trabalhando. O compilador irá apresentar um erro ao dividir dois números de ponto flutuante utilizando o operador inteiro **`div`** ou dois inteiros utilizando o operador **`/`**.

Operadores de Incremento e Decremento

Os operadores de incremento e decremento geram um código mais otimizado para a adição e subtração de variáveis inteiras. Os comando **`Inc()`** e **`Dec()`** realizam a adição e a subtração. Pode-se utilizar estes operadores como um ou dois parâmetros.

```
...
var
      VInt: Integer;
begin
      VInt := 10;
      Inc(VInt)    // Produz resultado idêntico a VInt := VInt + 1
      Inc(VInt, 3) // É idêntico a VInt := VInt + 3
      Dec(Vint, 5) // É idêntico a Vint := Vint - 5
end.
```

Tipos de Dados

A definição de tipos de dados corretos para as variáveis é muito importante, pois além de permitirem a economia de memória, eles tem, em muitos casos, de ter compatibilidade com os seus respectivos tipos de dados do Windows.

Tipos de Dados NUMÉRICOS INTEIROS

Tipo	Tamanho (em bytes)	Domínio
Byte	1	0..255
ShortInt	1	-128..127
Word	2	0..65535
SmallInt	2	-32768..32768
Longint	4	-2147483648..2147483647

Os tipos inteiros genéricos são **CARDINAL** e **INTEGER**. O tipo **Cardinal** representa um inteiro sem sinal e o tipo **Integer** representa um inteiro com sinal. O tamanho e formato do tipo **Integer** e do tipo **Cardina**l variam de acordo com as diferentes implementações da linguagem Object Pascal, mas geralmente são aqueles resultam em operações inteiras mais eficientes para uma determinada CPU ou Sistema Operacional.

Tipo	Tamanho (em bytes)	Domínio
Cardinal	2	-32768.. 32767
Integer	2	0.. 65636
Cardinal	4	0.. 2147483647
Integer	4	-2147483648.. 2147483647

Tipos de dados NUMÉRICOS REAIS e MOEDA

Tipo	Tamanho (em bytes)	Domínio
Single	4	$1.5 * 10^{-45}$ .. $3.4 * 10^{38}$
Real	6	$2.9 * 10^{-39}$ .. $1.7 * 10^{39}$
Double	8	$5.0 * 10^{-324}$ .. $1.7 * 10^{308}$
Extended	10	$3.4 * 10^{-4932}$ .. $1.1 * 10^{4392}$
Comp	8	-263^{10} .. 263^{-10}
Currency	8	-922337203685477.5808.. 922337203685477.5807

O tipo *Currency* é o tipo MOEDA. Ele tem precisão de quatro casas decimais e é compatível com os bancos de dados que representam dinheiro.

Tipos de dados BOOLEANOS

Tipo	Tamanho (em bytes)	Armazenamento
Boolean	1	1 byte
ByteBool	1	1 byte
WordBool	2	uma palavra
LongBool	4	uma palavra dupla

A existência de diversos tipos lógicos que armazenam a mesma coisa (TRUE ou FALSE) dá-se pela necessidade de compatibilização com os tipos do Windows.

Tipos de Dados Caracter

Tipo	Tamanho (em bytes)	Armazenamento
Char	1	1 caracter ANSI
AnsiChar	1	1 caracter ANSI
WideChar	2	1 caracter Unicode

- **AnsiChar**: É o padrão de caracteres ANSI de 1 byte que todos nós conhecemos.
- **WideChar**: É um tipo de 2 bytes surgido com o Delphi 2 para representar um caracter Unicode.
- **Char**: É idêntico ao AnsiChar.

Tipos de Dados String

Tipo	Tamanho (em bytes)	Armazenamento
ShortString	255	ANSIChar
AnsiString	até 3 Gb	ANSIChar
String	255 ou até 3 Gb	ANSIChar
WideString	até 1,5 Gb	WideChar

O tipo AnsiString é padrão. Quando é declarada uma variável como **`STRING`** é assumido AnsiString. Este tipo surgiu com o Delphi 2 para facilitar a manipulação de

strings que excediam o tamanho de 255 caracteres (ShortString). As variáveis declaradas com este tipo são alocadas dinamicamente e só ocupam memória enquanto estão sendo utilizadas e a medida que vão aumentando de tamanho, o seu espaço em memória é realocado automaticamente.

Tipo PChar

O tipo PChar permite também que você manipule strings. Quando você utiliza uma variável com o tipo PChar, o Delphi assume que todo o gerenciamento de alocação e liberação de memória será realizado pelo código do programa.

As variáveis do tipo PChar possuem como delimitador de fim de string o caracter especial #0.

Tipo Variant

O tipo de dados **variant** é um tipo único que pode manipular diferentes tipos de dados. O tipo variant pode manipular strings, inteiros, reais, entre outros. Porém uma variável do tipo variant não poderá manipular tipos de dados dinâmicos, como, por exemplo, os ponteiros (pointer) e as classes de dados (os objetos). Exemplo

```
...
Var
  V1, V2, V3: Variant;
Begin
  V1 := 1200;
  V2 := 'Borland Delphi';
  V3 := 3.1415;

  V1 := V1 + V2 + V3;

end.
```

Note que para o mesmo tipo foram atribuídos três diferentes tipos. Portanto, deve-se tomar cuidado quando realizar operações envolvendo este tipo de dado.

No exemplo acima, o conteúdo da variável V1 será igual a '1200Borland Delphi3.1415'.

Veja que neste caso o compilador irá converter todos as variáveis envolvidas na operação para o tipo String e realizar uma concatenação.

Tipos Definidos pelo Usuário

Número inteiros, strings e ponto flutuante freqüentemente não são suficientemente adequados para representar variáveis nos problemas do mundo real. Neste casos deve-se criar tipos complexos para melhor representar variáveis de um determinado problema. Em Pascal, os tipos definidos pelo usuário geralmente são da forma de registros (records) ou objetos.

Arrays

A linguagem Object Pascal permite criar arrays (matrizes) de qualquer tipo. Por exemplo, uma variável declarada como Array com oito números inteiros:

```
Var
  A: Array[0..7] of integer;
```

A declaração de arrays em Pascal possui uma propriedade que difere de outras linguagens: A array não precisa iniciar com um determinado número.

```
Var
  A: Array[28..30] of integer;
```

Records

O tipo record permite que se agrupe em uma única variável, um conjunto de tipos de dados diferentes. A aplicação deste tipo de dado é bastante semelhante ao conceito de registro, ou seja, ou seja, uma variável do tipo record é composta por vários campos.

`Record` é tipo estruturado em Pascal, equivalente ao **`struct`** do C.

```
{ Pascal }
Type
MyRec = record
          I:integer;
          D:double;
        end
```

```
/* C */
typedef struct {
                 int i;
                 double d;
                }MyRec;
```

Quando se trabalha com records, utiliza-se o caracter ponto para acessar cada campo.

```
...
var
  N: MyRec
Begin
  N.i := 23;
  N.d := 3.4;
end;
```

Pointers

O tipo de dado *Pointer* é uma variável que armazena um endereço de memória, e não um valor, como estamos acostumados a trabalhar. Um exemplo de variável do tipo pointer é o tipo PChar, mostrado anteriormente. Existem duas formas de implementação para o tipo pointer: pointer tipados e pointers não tipados.

Uma **variável pointer** é **tipada** quando se conhece o tipo de dado que está armazenado no endereço de memória contido na variável pointer.

```
...
var
  PP1: ^Integer;
begin
  New(PP1);      // Aloca memória para a variável PP1
  PP1^ := 1998; // Atribui valor ao endereço de memória
                // apontada pela variável PP1
  Dispose(PP1); // Libera memória que foi alocada para PP1
end.
```

Uma **variável pointer não-tipada** é uma variável que armazena um endereço de memória que não se conhece o tipo de conteúdo deste endereço.

```
...
var
  PP2: Pointer;
begin
  GetMem(PP2, 100);          //Aloca 100 bytes para PP2
  StrPCopy(PP2, 'Delphi 3') //Atribui valor ao endereço de
                          //memória apontado pela variável PP2
  FreeMem(PP2);           // Libera memória alocada para PP2
end.
```

É muito comum a utilização de variáveis do tipo record juntamente com o pointer para a implementação de listas (pilhas, filas, árvores, grafos, etc.).

Objetos

Pode-se dizer que um objeto é um **`record`** que contém *functions* e *procedures*.

```
type
TobjetoFilho =      class( TObjetoPai )
                            V: Integer;
                            procedure FazNada;
                    end;
```

Estruturas de Decisão

No Object Pascal existem dois comandos para tomar decisões durante a execução de um programa. São as instruções **`IF`** e **`CASE`**.

O comando IF

A instrução IF permite controlar a execução de outras instruções (ou bloco de instruções) do programa, dependendo do valor de uma variável ou expressão booleana. Quando a expressão booleana tem o valor True, a instrução (ou bloco) que segue a palavra THEN é executada. Quando a expressão booleana tem o valor False, a instrução (ou bloco) que segue a palavra ELSE é executada.

```
IF expressão booleana THEN instrução;
```

```
IF expressão booleana THEN
      instrução1
ELSE
      instrução2;
```

```
IF expressão booleana THEN
        BEGIN
           instrução;
           ...
           instrução;
        END
ELSE
        BEGIN
           instrução;
           ...
           instrução;
        END;
```

expressão booleana: constante, variável ou expressão boleana.

instrução: qualquer instrução Pascal, inclusive outra instrução IF.

- **Atenção**

Se uma instrução **if** possui várias condições (expressões booleanas) estas devem vir entre parêntesis.

```
if (x=7) and (y=8) then ...
```

O Comando CASE

O comando CASE é equivalente a uma série de comandos IF. Se a variável de controle é igual a uma das constantes de uma lista, a instrução ou bloco de instruções associada à lista é executada. Se a variável de controle não é igual a nenhuma das constantes, a declaração case não terá nenhum efeito e o computador passa para à instrução seguinte do programa.

```
CASE variável-controle OF
        constante1:  instrução1;
        constante2:  instrução2;
        ...
ELSE
        instrução1;
END;
```

variável-controle: é uma variável do tipo inteiro, caracter ou definido pelo usuário, mas não pode ser do tipo real.

constantes: pode ser um único valor, vários valores isolados separados por vírgula ou uma faixa de valores, indicadas pelo valor mínimo e máximo separados por dois pontos finais (..).

Estruturas de Repetição

Na linguagem Pascal estão disponíveis três estruturas de repetição que fornecem uma grande flexibilidade na representação de um processo repetitivo. Um loop executa uma instrução (ou bloco de instruções) repetidamente.

As três estruturas de repetição em Pascal são representadas pelas palavras reservadas FOR, WHILE e REPEAT.

O Comando FOR

O comando FOR é usado para repetir várias vezes a mesma instrução ou bloco de instruções. Durante a execução, uma variável usada como índice assume uma série de valores, desde um dado valor inicial até um valor final. Na forma FOR-TO, o índice assume valores crescentes; na forma FOR-DOWNTO, o índice assume valores decrescentes. A forma geral da declaração é a seguinte:

```
FOR índice := inicial TO final DO instrução
```

```
FOR índice := inicial TO final DO
BEGIN
       instrução1;
       instrução2;
       ...
END;
```

```
FOR índice := inicial DOWNTO final DO
BEGIN
       instrução1;
       instrução2;
       ...
END;
```

índice:	O *índice* é uma variável ordinal. O *índice* pode ser do tipo inteiro, caracter, byte ou definido pelo usuário. Não pode ser uma variável real.
inicial:	Valor inicial da variável índice.
final:	Valor final da variável índice.
instrução:	Qualquer instrução Pascal.

As variáveis ordinais possuem uma propriedade chamada "ordem". Em outra palavras, faz sentido falar em número inteiro "seguinte" ou no caracter "anterior". O número inteiro que vem depois de 6 é 7; o caracter que precede M é L. As variáveis reais **não** são ordinais. Não faz sentido falar no número real "seguinte", porque entre dois números reais quaisquer sempre existe outro número real.

É importante notar que se *inicial* for maior ou igual a *final*, as instruções contidas no loop do tipo FOR-TO serão simplesmente ignoradas. O mesmo acontecerá em um loop do tipo FOR-DOWNTO se *inicial* for menor que *final*.

- **Atenção**

O valor do *índice* só é definido no interior do loop. Depois que o programa sai de um loop **FOR-TO** ou **FOR-DOWNTO**, a variável usada como *índice* tem um valor indefinido. Se o valor dessa variável for usada depois de o programa sair do loop, o programa irá se comportar de forma imprevisível.

O Comando REPEAT

A instrução REPEAT faz com que uma instrução (ou bloco de instruções) seja executada repetidas vezes, até que uma certa condição seja satisfeita. Observe que na forma composta da declaração, as palavras BEGIN e END não são usadas para delimitar as declarações que fazem parte do loop; este sempre começa pela palavra REPEAT e termina na palavra UNTIL.

```
REPEAT instrução UNTIL condição;
```

```
REPEAT
      instrução1;
      instrução2;
      ...
UNTIL condição;
```

condição	é uma expressão booleana.
instrução	é qualquer instrução Pascal.

Como a *condição* é verificada no final do loop, todo loop do tipo REPEAT é executado pelo menos uma vez. Se a *condição* nunca assume o valor **True**, o loop só pára se o programa for interrompido do manualmente.

O Comando WHILE

O loop WHILE é semelhante ao loop REPEAT. A principal diferença está no fato de que o loop REPEAT é sempre executado pelo menos uma vez; o loop WHILE pode não ser executado nenhuma vez.

```
WHILE condição DO instrução.
```

```
WHILE condição DO
BEGIN
        instrução1;
        instrução2;
        ...
END;
```

condição	é uma expressão booleana.
instrução	é uma instrução Pascal. Na forma composta do loop WHILE não é preciso colocar um ponto e vírgula depois da última instrução.

Procedimentos e Funções

Procedimentos (**procedures**) e Funções (**functions**) são partes do programa que podem executar uma tarefa específica quando chamamos de um dos módulos do programa. Após a execução do procedimento ou função, o controle da execução retorna ao módulo do programa que executou a chamada.

Os procedimentos e funções são executados de forma bastante semelhante, porém, uma função retorna um valor de retorno, enquanto que um procedimento apenas executa um bloco de comandos.

```
Procedure ReajustaSalario;
begin
        if Salario > 600 then
                Salario := Salario * 1.05
        else
                Salario := Salario * 1.10;
end;
```

```
Function HouveReajuste: Boolean
begin
       if Salario > 600 then
              result := false
       else
              result := true;
end;
```

Note que na implementação da função é necessário indicar o tipo de retorno da função. Para a função retornar o valor desejado, deve-se utilizar a variável de sistema **Result**.

```
begin
  while Salario < 600 do
    ReajustaSalario;
End;
```

```
begin
  if HouveReajuste then
    ShowMessage('Reajuste');
end;
```

Procedimentos e Funções com Parâmetros

A linguagem Pascal permite se utilize parâmetros na implementação de procedimentos e funções. O Pascal permite passagem de parâmetros por valor ou por referência. Os parâmetros passados podem ter um tipo básico ou um tipo definido pelo usuário ou *arrays abertas* (que será visto mais adiante). Um parâmetro pode também ser constante, caso seu valor não seja alterado durante a execução da procedure ou da function.

Passagem de Parâmetros por Valor

A passagem de parâmetros por valor é a forma padrão do Pascal. Quando um parâmetro é passado por valor, significa que uma copia local daquela variável é criada e a function ou procedure opera sobre essa cópia.

```
...

var
   m: String;

procedure Mens(s: string );
begin
   s := 'Mensagem: '+ s;
   ShowMessage( s );
end;

begin         // Inicio do Programa Principal

   m := 'teste de passagem de parametros';
   Mens( m );

end.
```

No exemplo acima, na chamada da **procedure Mens**, apenas o valor contido na variável **m** é passado para a variável **s** da procedure. Desta forma, a **procedure** irá operar com a variável **s**, sem afetar o conteúdo da variável **m**.

Passagem de Parâmetros por Referência

Passar um parâmetro por referência significa que a função ou procedimento poderá alterar o valor da variável enviada na sua chamada. Para passar uma variável por referência utiliza a palavra **var** na lista de parâmetros da *procedure* ou *function.*

```
procedure ChangeMe(var x: LongInt);
begin
       x := x + 2;
end;

begin

   z := 10
   ChangeMe( z );

end.
```

No exemplo acima, o endereço de memória da variável **x** será o mesmo endereço da variável **z**, utilizada na chamada da procedure. Desta forma, toda a alteração realizada na variável **x** será refletida na variável **z**. Assim sendo, após a chamada da **procedure ChangeMe(z)** a variável **z**, que inicialmente continha o valor 10, terá o valor 12.

Parâmetros Constantes

Se o valor de um parâmetro não vai ser alterado durante a execução da procedure ou function, pode-se declara-lo como constante através da cláusula **`conts`**. A palavra **`const`** previne qualquer modificação de valor e também gera um código otimizado.

```
procedure ApresentaMensagem(const Mens: String);
```

Parâmetros com Arrays Abertas

Através deste tipo de parâmetro é possível passar um número variável de argumentos para procedures ou functions. Pode-se passar arrays cujos elementos possuem o mesmo tipo ou pode-se passar arrays constantes de tipos diferentes. O exemplo abaixo declara uma função que recebe uma array aberta com elementos inteiros.

```
function TesteArray(A: Array of integer): integer;
```

Pode-se passar variáveis, constantes ou expressões para a função acima.

```
var
      i, Rez: Integer;
const
      j = 23
begin
      i := 8;
      Rez := TesteArray( [ i, 50, j, 89 ] );
      ...
```

Para se trabalhar com uma array aberta dentro da procedure ou function pode-se utilizar as funções **`High()`**, **`Low()`** e **`SizeOf()`** para se obter informações sobre a array.

A linguagem Object Pascal suporta também um array de constantes que permite passar dados de tipos diferentes para uma function ou procedure.

```
procedure TesteArrayConstante( A: array of const );
```

A função acima pode ser chamada da seguinte maneira:

```
TesteArrayConstante(['Maceio',90,5.6,@X,3.14159,True,'s']);
```

Units

As Units são módulos de programa onde são agrupados os procedimentos e funções que compõem a aplicação Delphi. Uma Unit é composta de áreas pré-definidas.

```
unit Unit1;

interface

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

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

var
  Form1: TForm1;

implementation

{$R *.DFM}

end.
```

A Área Unit

Nesta área encontra-se o início da Unit, determinada pela palavra **Unit**, seguida pelo seu identificador (nome). O nome da unit será gerado pelo Delphi e terá o mesmo nome que o arquivo armazenado em disco. Isto é, inicialmente o Delphi gera automaticamente o nome da Unit. Ao ser gravado no disco, a Unit assume o nome dado ao arquivo.

A Área Interface

A área de interface encontra-se sempre após a área da Unit e antes da área de implementation. Esta área é utilizada para realizar as declarações de variáveis, constantes, tipos, procedimentos e funções, etc. As declarações posicionadas nesta área poderão ser compartilhadas por outras Units ou pelo módulo principal da aplicação.

Nesta área somente poderão aparecer declarações, nunca aparecerá linhas de comando.

A Área Implementation

Nesta área será encontrado basicamente a implementação dos procedimentos e funções. Pode ainda conter declarações de variáveis, constantes, tipos, etc. que não poderão ser compartilhados por outras Units, nem pelo próprio módulo principal da aplicação. Isto e', os dados declarados na área *implementation* são dados locais, somente visíveis dentro da Unit na qual foram declarados.

Áreas Opcionais

A Área Initialization

Esta área, quando existente, deve conter linhas de comando responsáveis pela inicialização de variáveis, objetos, etc., declarados dentro da Unit. A execução desta área é realizada antes da execução do módulo principal da aplicação e ocorre uma única vez.

A Área Finalization

Esta área deve aparecer após a área Initialization e deve conter linhas de comandos relativas a desalocação de memória de variáveis do tipo Pointer ou Objetos. A execução desta área é realizada s sua aplicação está sendo encerrada.

A cláusula Uses

A cláusula **uses** serve para que possamos tornar acessíveis as variáveis, constantes, procedimentos e funções declarados na área **Interface** de uma determinada **Unit**. A cláusula uses deve aparecer dentro da Unit na qual deseja-se manipular dados declarados fora do se escopo.

```
program Prog;
  uses Unit2;
  const a = b;
begin
end.
```

```
unit Unit2;
  interface
    uses Unit1;
    const b = c;
  implementation
end.
```

```
unit Unit1;
  interface
    const c = 1;
  implementation
    const d = 2;
end.
```

Referência Circular

A cláusula **uses** pode aparecer na área **interface** ou na área **implementation**, sem interferir em desempenho ou tamanho da aplicação. Porém, existe um caso em que se torna obrigatório a declaração da cláusula uses dentro da área **implementation** da Unit. Isto acontece quando duas Units fazem referência entre si. Este processo é conhecido como **Referência Circular**.

```
Unit UnAcessa

Interface
  var B: Integer;

Implementation
  uses UnDeclara;

procedure Inicia;
begin
  A := 10;
end;

end.
```

⟶

⟵

```
Unit UnDeclara

Interface
  var A: Integer;

Implementation
  uses UnAcessa;

procedure Reset;
begin
  B := 0;
end;

end.
```

No exemplo acima mostra um caso de referência circular que foi solucionada devido à declaração da cláusula **uses** ter sido feita na área **implementation**. Caso a declaração da cláusula **uses** tivesse sido feita na área **interface**, seria apresentado um erro durante a compilação

Exceções

Exceções são erros que ocorrem durante a execução do programa. Existem diversos **tipos de exceções**, que podem ocorrer nas seguintes operações:

- alocação de memória;
- criação e utilização de objetos;
- cálculo de expressões matemáticas;
- manipulação de arquivos;
- utilização de recursos do sistema operacional.

Algumas das principais características das exceções no DELPHI são:

- a instância da exceção fornece informações sobre o tipo de problema ocorrido;
- usuário pode definir suas próprias exceções, utilizando a classe ***Exception***;
- para forçar a execução de uma exceção utiliza-se a cláusula ***Raise***;
- os tipos de exceção normalmente devem iniciar com a letra **E**.
- uma exceção gera um evento denominado ***OnException***.

As exceções podem ser tratadas através das instruções:

- **Try ... Except**
- **Try ... Finally**

Try ... Except

O bloco ***try ... except*** é utilizado para definir um código de programa a ser executado no momento que ocorrer uma exceção.

```
try
   { Bloco de comandos}
except
   { comandos executados quando ocorrer uma exceção}
end;
```

Exemplo: tratamento de uma divisão por zero.

```
...
Valor1 := StrToInt(Edit1.Text);
Valor2 := StrToInt(Edit2.Text)
try
      Resultado := Valor1 div Valor2;
except
      ShowMessage ('Divisão por Zero');
end;
...
```

No entanto, pode ocorrer mais de uma exceção dentro de um mesmo bloco. Neste caso é necessário utilizar o comando **on ... do**.

On ... Do

O comando ***on ... do*** permite tratar mais de um tipo de exceção de um bloco **try ... except**.

```
on
   {Tipo de exceção}
do
   {Bloco de comandos da exceção}
```

Exemplo: tratamento de uma divisão por zero e da digitação de um caracter em um campo que deveria ser numérico.

```
...
try
   Valor1 := StrToInt (Edit1.Text);
   Valor2 := StrToInt (Edit2.Text)
   Resultado := Valor1 div Valor2;
except
   on EDivByZero do
      ShowMessage ('Divisão por Zero');
   on EInOutError do
      ShowMessage ('Os valores devem ser numéricos');
end;
```

Tipos de Exceção

Os principais tipos de exceção da RTL (*RunTime Library*) do DELPHI, a serem tratadas nos blocos **on ... do** são:

Nome	Descrição
EaccessViolation	Ocorre quando se tenta acessar uma região de memória inválida (ex: tentar atribuir valor a um ponteiro cujo conteúdo é nil).
EconvertError	ocorre quando se tenta converter um string em um valor numérico (ex: utilizar a função StrToInt em uma letra).
EdivByZero	ocorre na divisão de um número por zero.
EinOutError	ocorre numa operação incorreta de I/O (ex: abrir um arquivo que não existe).
EintOverFlow	ocorre quando o resultado de um cálculo excedeu a capacidade do registrador alocado para ele (para variáveis inteiras).
EinvalidCast	ocorre quando se tenta realizar uma operação inválida com o operador as (ex: tentar usar um Sender com uma classe que não corresponde a seu tipo).
EinvalidOp	ocorre quando se detecta uma operação incorreta de ponto flutuante.
EinvalidPointer	ocorre quando se executa uma operação invalida com um ponteiro (ex: tentar liberar um ponteiro duas vezes).
EoutOfMemory	ocorre quando se tenta alocar memória mas já não existe mais espaço suficiente.
EoverFlow	ocorre quando o resultado de um cálculo excedeu a capacidade do registrador alocado para ele (para variáveis de ponto flutuante).
ErangeError	ocorre quando uma expressão excede os limites para a qual foi definida (ex: tentar atribuir 11 ao índice de um vetor que pode ir no máximo até 10).
EstackOverflow	ocorre quando o sistema não tem mais como alocar espaço de memória na Stack.
Eunderflow	ocorre quando o resultado de um cálculo é pequeno demais para ser representado como ponto flutuante.
EzeroDivide	ocorre quando se tenta dividir um valor de ponto flutuante por zero.

Try ... Finally

O bloco ***try ... finally*** permite que um bloco de comandos seja executado, mesmo quando houver uma exceção.

```
try
{ Bloco de comandos}
finally
{Comandos executados mesmo quando ocorrer uma exceção}
end;
```

Por exemplo, garantir que um arquivo será fechado mesmo quando ocorrer uma exceção.

```
...
assign (arq, 'teste.txt');
try
  reset(arq)
  // outros comandos sobre o arquivo
finally
  close (arq)
end;
...
```

Universidade Federal da Paraíba
Centro de Ciências Sociais Aplicadas
Departamento de Ciência da Informação
Programa de Pós-Graduação em Ciência da Informação
Ação para cidadania e acesso livre à informação

Ambiente Integrado de Desenvolvimento

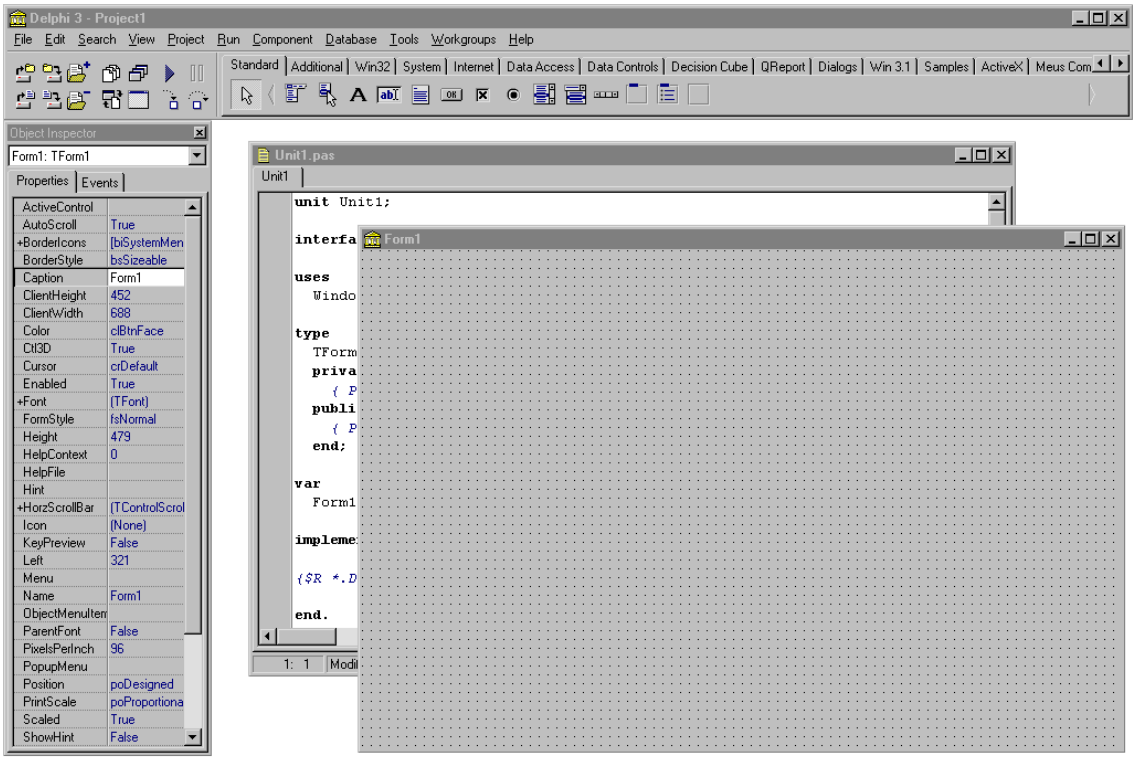
– IDE –

João Pessoa, Maio de 2011

3

O Ambiente DELPHI

O ambiente de desenvolvimento do Delphi possui as características **IDE** (*Integrated Developer Environment* – Ambiente Integrado de Desenvolvimento).



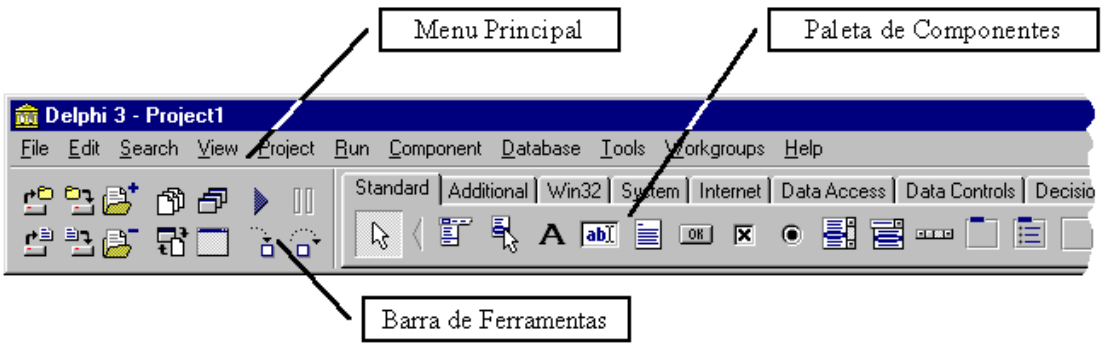
O DELPHI possui um conjunto de ferramentas que permitem facilitar e agilizar a construção de programas, permitindo uma melhor interação entre o programador e o computador. Seus principais componentes são:

- Janela Principal
- Janela *OBJECT INSPECTOR*
- Janela *FORM DESIGNER*
- Janela *CODE EDITOR*

Janela Principal

A janela principal controla o funcionamento do Delphi. É através dessa janela que mantemos o Delphi aberto, que identificamos qual projeto está ativo, que executamos os comandos de compilação, entre outras funcionalidades. Esta janela pode ser dividida em três partes:

- **Menu Principal**: contém as opções de utilização do DELPHI;
- **Barra de Ferramentas**: contém botões que agilizam determinadas funções do DELPHI;
- **Paleta de Componentes**: contém diversas páginas de componentes que podem ser utilizados na construção do projeto.



A Janela Principal: dividida em três partes.

Menu Principal

Assim como toda aplicação Windows, no menu principal estão presentes os comandos de abertura, criação, compilação, configuração do ambiente e chamadas a ferramentas auxiliares como, por exemplo, o **Image Editor**.

❑ File

Abrir, salvar, fechar e imprimir projetos e arquivos, e acrescentar novos formulários ou **Units** ao projeto que está aberto.

Comando	Descrição
New	Abre a caixa de diálogo **New Items** que contém novos itens que podem ser criados.
New Application	Cria um novo projeto contendo um novo formulário, uma **Unit** e o arquivo **.DPR.**
New Form	Cria e adiciona um novo formulário ao projeto.
New Data Module	Cria e adiciona um novo formulário do tipo **DataModule** ao projeto.
Open	Abre uma caixa de diálogo para carregar um projeto, um formulário ou uma **Unit** no **Code Editor**.
Reopen	Apresenta um menu contendo uma lista dos últimos projetos e módulos utilizados.
Save	Grava o arquivo ativo com o seu nome atual.
Save As	Grava o arquivo ativo com um novo nome.
Save Project As	Grava o projeto corrente com um novo nome.
Save All	Grava todos os arquivos abertos, tanto o projeto como seus módulos.
Close	Fecha uma **Unit** e/ou Formulário ativo.
Close All	Fecha todos os arquivos.
Use Unit	Adiciona uma **Unit** à cláusula **uses** do módulo ativo.
Add to Project	Adiciona um arquivo (**Unit**/Formulário) ao projeto.
Remove From Project	Remove um arquivo do projeto.
Print	Envia o arquivo ativo para a impressora.
Exit	Fecha o projeto aberto e finaliza a execução do Delphi.

❑ Edit

Comandos para manipular textos e componentes em tempo de projeto.

Comando	Descrição
Undelete (Undo)	Desfaz a última ação efetuada.
Redo	Reverte um comando **Undo.**
Cut	Remove um item selecionado e o coloca no **Clipboard.**
Copy	Coloca uma copia do item selecionado no **Clipboard**, permanecendo o item original.
Paste	Copia o conteúdo do **Clipboard** na janela **Code Editor** ou no formulário.
Delete	Remove o item selecionado.
Select All	Seleciona todos os componentes no formulário.
Align To Grid	Alinha o componente selecionado à grade do formulário.
Bring To Front	Move o componente selecionado para a frente de outro componente.
Send To Back	Move o componente selecionado para trás de outro componente.
Align	Alinha os componentes selecionados.
Size	Altera o tamanho dos componentes selecionado.
Scale	Altera o tamanho de todos os componentes do formulário.
Tab Order	Altera a ordem (tabulação) dos componentes do formulário ativo.
Creation Order	Modifica a ordem em que os componentes não visuais serão criados.
Lock Controls	Quando acionado, não permite que componentes sejam movidos (arrastados) de suas posições
Add to interface	Define um novo método, evento ou propriedade para um componente **ActiveX**

❑ Search

Localiza texto, erros, objetos, units, variáveis e símbolos no **Code Editor**.

Comando	Descrição
Find	Procura por um texto especificado e marca a primeira ocorrência encontrada no **Code Editor**
Find In File	Procura por um texto especificado e apresenta cada ocorrência em uma janela abaixo do **Code Editor**.
Replace	Procura por um texto especificado e o substitui por um novo texto.
Search Again	Repete a procura.
Incremental Search	Procura por um texto movendo o cursor diretamente à próxima ocorrência do texto digitado na barra de status.
Go to Line Number	Move o cursor para uma determinada linha.
Find Error	Procura pelo erro de execução mais recente.
Browse Symbol	Procura por um determinado símbolo.

- **View**

Apresenta (ou esconde) diferentes elementos do ambiente Delphi e abre janelas do depurador de código (*debugger*).

Comando	Descrição
Project Manager	Mostra a janela do **Project Manager.**
Project Source	Apresenta o arquivo de projeto (.DPR) no **Code Editor.**
Object Inspector	Mostra o **Object Inspector.**
Alignment Palette	Mostra uma janela contendo opções para alinhamento de componentes no formulário (**Alignment Palette**).
Browser	Mostra o **Object Browser.**
Breakpoints	Apresenta uma janela contendo uma lista dos **breakpoints** atualmente definidos (**Breakpoints List**).
Call Stack	Apresenta uma lista da seqüência atual das rotinas chamadas por um programa (**Call Stack**).
Watches	Apresenta uma janela para visualizar o conteúdo de variáveis durante a execução de um programa (**Watch List**).
Threads	Apresenta um janela de status dos **threads** (**Thread Status**).
Modules	Mostra uma janela contendo os módulos da aplicação. (**Modules**).
Component List	Apresenta uma lista de todos os componentes (**Components**).
Window List	Apresenta uma lista das janelas abertas (**Window list**).
Toggle Form/Unit	Alterna entre o formulário e sua **Unit** correspondente.
Units	Mostra as **Units** que fazem parte do projeto.
Forms	Mostra os formulários que fazem parte do projeto.
Type Library	Mostra a janela **Type Library** que permite examinar e criar informações de tipo para controles **ActiveX**.
New Edit Window	Abre uma nova janela para o **Code Editor.**
SpeedBar	Esconde/mostra a Barra de Ferramentas (**SpeedBar).**
Component Palette	Esconde/Mostra a Paleta de Componentes.

❑ Project

Permite adicionar ou remover novos módulos em um projeto, bem como compilá-lo;

Comando	Descrição
Add to Project	Adiciona um arquivo ao projeto.
Remove from Project	Remove um arquivo do projeto.
Import Type Library	Importa uma biblioteca de tipos para um projeto.
Add To Repository	Adiciona um projeto ao **Object Repository**.
Compile	Compila todos módulos que foram alterados desde a última compilação.
Build All	Compila todos os arquivos do projeto.
Syntax Check	Compila o projeto sem passar pelo processo de *link-edição*.
Information	Mostra informações sobre o projeto
Web Deployment Options	Configura um controle **ActiveX** ou **ActiveForm** para ser efetivamente utilizado em seu servidor Web.
Web Deploy	Após configurar e compilar o projeto, utilize este comando para que o controle ActiveX ou ActiveForm seja utilizado pelo servidor Web.
Options	Apresenta a janela de opções do projeto (**Project Options**).

❑ Run

Apresenta opções de execução e depuração do projeto.

Comando	Descrição
Run	Compila e executa a aplicação.
Parameters	Especifica parâmetros de inicialização da aplicação.
Register ActiveX	Registra o projeto no Windows. Disponível Quando for um projeto **ActiveX**.
Unregister ActiveX Servers	Remove o projeto do registro Windows. Disponível quando for um projeto **ActiveX**.
Step Over	Executa um programa linha por linha, sem passar pelas *procedures* ou *functions*.
Trace Into	Executa um programa linha por linha, passando inclusive pelas *procedures* e *funcitons*.
Trace To Next Source Line	Executa um programa parando a cada próxima linha executável.
Run To Cursor	Executa a aplicação a partir da localização do cursor na janela **Code Editor**.
Show Execution Point	Posiciona o cursor no ponto de execução.
Program Pause	Suspende temporariamente a execução de um programas.
Program Reset	Finaliza a execução do programa e libera a memória.
Add Watch	Abre a janela **Watch Properties** onde podem ser definidas as variáveis que serão visualizadas durante a execução do programa.
Add Breakpoint	Abre o **Edit Breakpoint** onde podem ser definidos **breakpoints** (pontos de parada do programa).
Evaluate/Modify	Abre a janela **Evaluate/Modify** onde pode-se calcular ou alterar o valor de uma expressão.

❑ Component

Permite a criação ou instalação de novos componentes no DELPHI.

Comandos	Descrição
New Component	Abre o **Component Expert** para criar uma **Unit** básica para um novo componente.
Install Component	Instala um componente em um **package** (pacote) novo ou já existente.
Import ActiveX Control	Adiciona bibliotecas de controles **ActiveX** em um projeto Delphi.
Create Component Template	Personaliza um componente e salva-o na Paleta de Componentes
Install Packages	Especifica os **packages** (pacotes) requeridos pelo projeto.
Configure Palette	Abre a janela de configuração de componentes e paletas.

❑ Database

Possui ferramentas que permitem criar, modificar e visualizar Consultas e Tabelas de Banco de Dados.

Comandos	Descrição
Explore	Abre o **Database Explorer** ou o **SQL Explorer**, dependendo da versão do Delphi. Ambos permitem criar, visualizar e alterar dados e **Alias**.
SQL Monitor	O **SQL Monitor** permite visualizar chamadas feitas através de ligações SQL (**SQL Links**) à um servidor remoto através do ODBC.
Form Wizard	O **Form Wizard** permite gerar facilmente formulários que apresentam dados de um banco de dados InterBase, Paradox, dBASE, ou Oracle.

❑ Tools

Permite visualizar e alterar configurações de ambiente, modificar a lista de programas externos ao Delphi.

Comandos	Descrição
Environment Options	Configura o ambiente e personaliza a aparência da Paleta de Componentes.
Repository	Apresenta a janela **Object Repository** para a configuração das páginas de componentes apresentados no comando **New** do menu **File**.
Configure Tools	Apresenta a janela **Tools Options** para adicionar ou remover programas do menu Tools.
Package Collection Editor	Cria e edita coleções de **packages** (pacotes). As coleções de pacotes são uma forma conveniente de agrupar pacotes para distribuição a outros programadores.
Image Editor	Cria e edita arquivos de **resource** (.RES), icones, bitmaps, e cursores para serem usados em aplicações Delphi.
Database Desktop	Cria, visualiza, modifica e consulta tabelas em formato Paradox, dBASE, e SQL.

❑ Workgroups

Configura o controle de versões em projetos envolvendo vários programadores. O controle de versões do Delphi é feito pelo *Intersolv PVCS Version Manager 5.1*.

Comandos	Descrição
Browse PVCS Projects	Apresenta a janela **PVCS Project** na qual pode ser feita todas as tarefas referentes ao controle de versões.
Manage Archive Directories	Quando a janela **PVCS** está aberta, apresenta a janela **Archive Directories** que permite adicionar, remover ou criar diretórios para o **PVCS Version Control** armazenar informações que compõe um projeto.
Add Project to Version Control	Permite salvar o projeto corrente em um diretório de controle de versão.
Set Data Directories	Apresenta a janela **Data Directories** para configurar a localização de diretórios públicos e privados.

- **Help**

Ajuda do DELPHI.

Comando	Descrição
Contents	Selecione uma das páginas (Conteúdo, Indice ou Localizar). A página **Conteúdo** apresenta uma lista de tópicos organizada hierarquicamente. A página **Índice** apresenta os tópicos listados em ordem alfabética. A página **Localizar** procura um texto especificado.
Keyword Search	Abre a janela **Help** posicionada na opção **Índice**, descrita anteriormente.
What's New	Apresenta um resumo as novas características e extensões do Delphi 3.0 em relação às versões anteriores.
Getting Started	Fornece informações sobre as principais características do ambiente Delphi e os fundamentos de programação.
Using Object Pascal	Descreve a sintaxe e a semântica da linguagem de programação **Object Pascal**.
Developing Applications	Fornece informações para tarefas avançadas. Fala sobre aplicações de banco de dados, componentes personalizados e aplicações Internet e Intranet. Explica como trabalhar com **COM** e **ActiveX**.
Object and Component Reference	Referência completa da VCL, tipos, rotinas, variáveis e constantes.
Borland Home Page	Abre seu Web Browser e posiciona no *site* da Borland.
Delphi Home Page	Ligação direta para o *site* da Borland que se refere ao Delphi
Borland Programs and Services	Ligação a *sites* de programas e serviços. Fornece informações sobre suporte técnico.
About…	Apresenta os direitos autorais e informação sobre a versão utilizada. Possui também ligação para o *site* da Borland.

Barra de Ferramentas

A barra de ferramentas contém as principais opções do menu principal e é utilizada para acessar os comandos de forma mais rápida.

O conjunto de botões da barra de ferramentas pode ser personalizada pressionando-se o botão direito do mouse sobre ela e escolhendo-se a opção **Properties** no menu.

Paleta de Componentes

Componentes são programas que representam os botões, os ComboBoxes, as tabelas. Etc. Cada página apresenta um conjunto de ícones que representam as classes cujos objetos que serão utilizados nos formulários das aplicações.

Para adicionar componentes em um formulário basta seleciona o componente e em seguida posiciona-lo no formulário.

Os componentes podem ser Visuais ou Não-Visuais. Os componentes visuais são aqueles que podem ser redimensionados e aparecem no formulário quando o projeto está sendo executado.

Os componentes Não-Visuais são visíveis apenas em tempo de edição e não podem ser redimensionados.

Todo componente Delphi é um objeto que pertence a uma classe pré-definida. Em sua maioria um componente possui **Propriedades**, **Métodos** e **Eventos**.

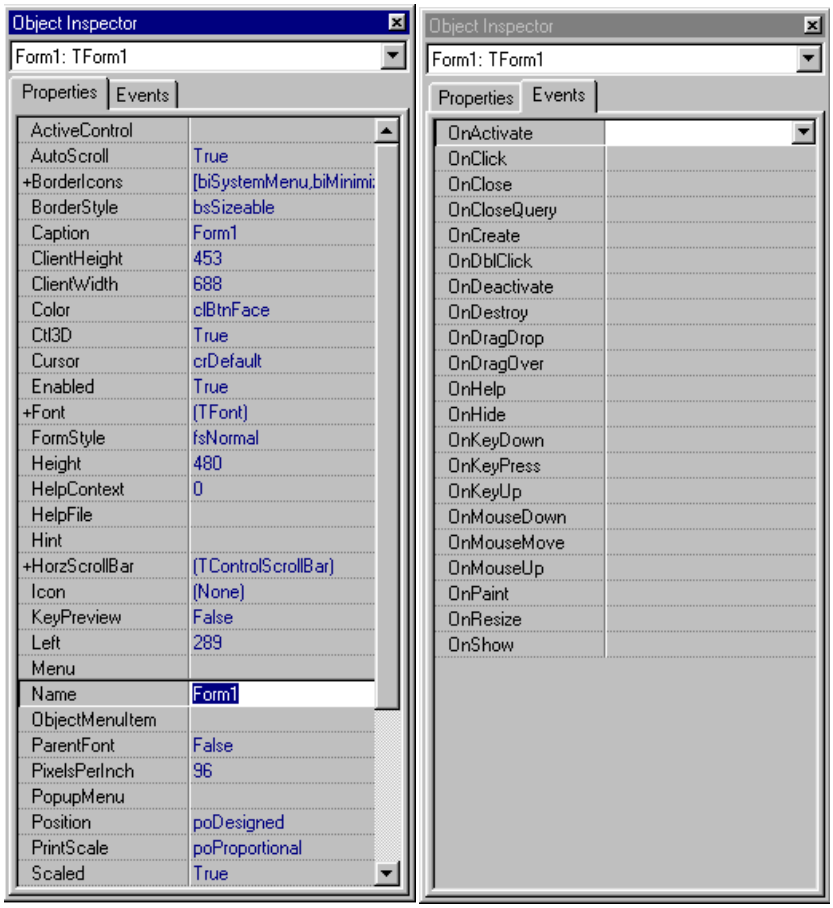
Propriedades são dados relativos a um determinado componente, e que determinam suas características visuais e seu estado. Por exemplo, o tamanho de um botão, se uma tabela está aberta ou fechada.

Eventos são trechos de programas (**procedures**) que são executados em resposta a uma determinada ação. Essa ação pode ser o clique do mouse sobre um botão da interface, a exclusão de um registro em uma tabela, o fechamento de um formulário, etc.

Métodos são procedimentos ou funções que executam uma determinada operação. Os métodos estão associados a um determinado objeto e são declarados na classe que este objeto pertence. Por exemplo, existem métodos para abrir tabelas, fechar formulários, excluir registros, etc.

O Object Inspector

A janela Object Inspector contém propriedades e eventos dos componentes inseridos em um formulário e também do próprio formulário. A janela do Object Inspector possui um ComboBox e está dividida em duas páginas.



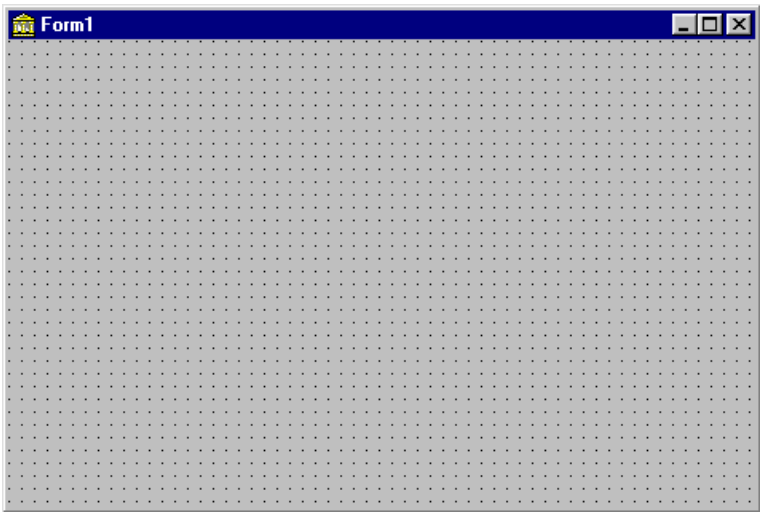
A Janela OBJECT INSPECTOR: propriedades e eventos

A página *Properties* (**Propriedades**) permite que se altere os valores das propriedades dos componentes contidos no formulário ou ainda as propriedades do próprio formulário. Já na página *Events* (**Eventos**) estabelecem-se ações a serem tomadas pelo componente a partir de um evento associado ao mouse, teclado, sistema operacional, etc.

O ComboBox existente no Object Inspector mostra o componente atualmente selecionado no formulário e apresenta uma lista com todos os componentes existentes no formulário.

O Form Designer

Um formulário (FORM) é a janela onde o desenvolvedor constrói sua aplicação. A partir de um formulário é que se estabelece a interação usuário-computador, através de botões, rótulos e outros componentes, estabelecendo-se funções, métodos ou eventos que serão ativados. Os componentes são dispostos na área útil do FORM.



A Janela FORM: interface.

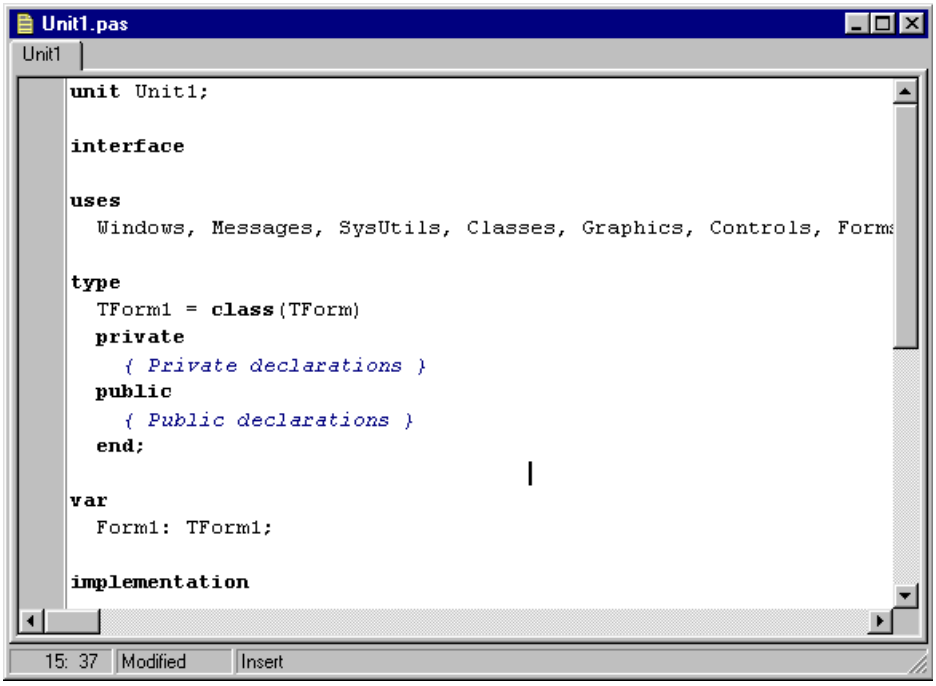
Existem quatro tipos de formulários:

- **Normal**: é o FORM padrão;
- **MDIForm:** os FORMS do tipo MDI (Multiple Document Interface), podem conter outros FORMS do tipo ***MDIChild***;
- **MDIChild:** são FORMS que sempre estão subordinados a um FORM ***MDIForm***;
- **StayOnTop:** são FORMS que sempre ficam visíveis na tela.

O Code Editor

A janela Code Editor (Editor de Código), é onde se desenvolve o programa fonte. É neste editor que se encontra a estrutura sintática propriamente dita da Linguagem. Cabe ressaltar, no entanto, que boa parte do código escrito é gerado automaticamente.

No topo do editor de código encontra-se um controle de múltiplas páginas, onde cada página corresponde a um módulo do programa (Unit). Geralmente as Units estão relacionadas aos formulários da aplicação. A cada vez que se adiciona um novo formulário na aplicação, o Delphi adiciona também uma nova página ao editor de código.



A Janela CODE EDITOR: código fonte

4

Biblioteca de Classes

Por ser uma linguagem de programação orientada a objetos o DELPHI possui uma **biblioteca de classes** denominada de **VCL (Visual Component Library)**, para auxiliar na construção de programas.

A biblioteca de classes do DELPHI pode ser visualizada no Browser (comando ***View*|*Browser*** do menu principal).

No DELPHI existem as classes ***visíveis*** e as ***não visíveis***.

As **classes não-visíveis** oferecem controles especializados que não são possíveis ou necessários através da interface visual. As classes não visíveis mais comuns são a **TApplication**, a **TScreen** e a **TPrinter**.

Classes Não-Visíveis

TApplication

Quando se inicia um projeto em DELPHI, automaticamente é criada uma variável *Application* que é do tipo *TApplication*. Esta classe permite a interação de sua aplicação com o Windows.

A atribuição de valores às propriedades do *TApplication* pode ser feita apenas em tempo de execução, ou durante o desenvolvimento do projeto na opção do menu *Project - Options*.

Os principais **métodos** da classe *TApplication* são:

- **Run()**: faz com que a aplicação seja executada. Este comando é colocado automaticamente no código fonte do projeto (veja em *View - Project Source*).
- **HelpCommand()**: carrega o arquivo de *help* da aplicação.

Algumas das principais **propriedades** da classe *TApplication* são:

- **MainForm**: contém o formulário principal da aplicação;
- **Title**: contém o título da aplicação que será apresentado para quando estiver minimizada;
- **HelpFile**: armazena o nome do arquivo de *Help* da aplicação;
- **Icon**: contém o ícone que representa a aplicação.

TScreen

Quando se inicia um projeto em DELPHI, automaticamente é criada uma variável *Screen* que é do tipo *TScreen*. Esta classe contém as características da tela na qual a aplicação está rodando.

Algumas das principais **propriedades** da classe *TScreen* são:

- **Cursor** : contém o cursor que está em uso;
- **Cursors**: contém a lista dos cursores disponíveis para a aplicação;
- **Fonts**: contém a lista de fontes disponíveis para a aplicação;
- **Forms**: contém a lista de formulários da aplicação;
- **Height**: contém a altura da tela (em pixels);
- **Width**: contém a largura da tela (em pixels).

TPrinter

Quando se inicia um projeto em DELPHI, automaticamente é criada uma variável *Printer* que é do tipo *TPrinter*. Esta classe contém as características da impressora que está sendo utilizada pelo Windows, além de permitir a impressão.

Os principais **métodos** da classe *TPrinter* são:

- **BeginDoc()**: inicia um processo de impressão.
- **EndDoc()**: finaliza um processo de impressão.
- **Abort()**: aborta o processo de impressão.
- **NewPage()**: inicia a impressão em uma nova página.

Algumas das principais **propriedades** da classe *TPrinter* são:

- **Printers**: contém a lista de impressoras instaladas no Windows.
- **PrinterIndex**: contém o índice da impressora ativada.
- **Orientation**: contém a orientação do papel.
- **PageHeight**: contém a altura da página de impressão.
- **PageWidth**: contém a largura da página de impressão.
- **PageNumber**: contém o número da página que está sendo impressa.
- **Title**: contém o título da página impressa.

Para a impressão de dados utilizando o *TPrinter* é necessário utilizar a classe **TCanvas**.

Componentes

As **classes visíveis** são aquelas que podem ser manipuladas através da Paleta de Componentes, ou seja, são os próprios componentes

A paleta de componentes é a biblioteca de classes que fornece recursos para o desenvolvimento visual em DELPHI. Os componentes VCL (***Visual Component Library***) representadas na Paleta de Componentes estão separados em páginas.

Os componentes visuais podem ter sua forma e tamanho alteradas no formulário, além das propriedades e eventos que podem ser alterados no **Object Inspector**. Eles aparecem durante a execução do aplicativo exatamente como foram definidos durante a construção do formulário.

STANDARD

A página Standard possui os componentes de interface mais comuns para a construção de aplicações Windows95/NT.

Página Standard: componentes mais comuns

MainMenu

Cria uma barra de menus em um formulário. Para definir as opções do menu, adicione o componente MainMenu no formulário e dê um duplo-click sobre ele para abrir o **Menu Designer**.

PopupMenu

Cria um menu popup que aparece quando o usuário clica o botão direito do mouse. Para definir as opções do menu e, adicione o componente PopupMenu no formulário e dê um duplo-click sobre ele para abrir o **Menu Designer**.

Label

Apresenta um texto que o usuário não poderá selecionar ou manipular.

Edit

Apresenta um campo de edição onde o usuário poderá digitar uma única linha de texto.

Memo

Apresenta uma área de edição onde o usuário poderá digitar diversas linha de texto.

Button

Cria um botão que o usuário escolhe para iniciar uma determinada ação.

CheckBox

Representa uma informação do tipo Sim/Não. Geralmente é utilizado para representar um grupo de opções que não são mutuamente exclusivas, podendo-se selecionar mais de uma opção de um mesmo grupo.

RadioButton

Apresenta uma informação do tipo Sim/Não. É geralmente utilizado para representar um grupo de opções mutuamente exclusivas, podendo o usuário escolher apenas uma opção de um mesmo grupo.

ListBox

Apresenta uma lista de opções.

ComboBox

Apresenta um campo de edição onde o usuário digita informações diretamente em uma linha de texto, ou escolher a partir de uma lista previamente definida.

ScrollBar

Fornece uma forma de alterar a área visível de uma lista ou um formulário.

GroupBox

Permite definir uma área para criar um grupo de componentes relacionados.

RadioGroup

Permite criar uma área contendo um agrupamento de componentes RadioButton.

Panel

Permite cria uma área que pode-se agrupar diversos tipos de componentes. Pode-se ainda utilizar o Panel para criar barra de ferramentas ou linhas de status.

ADDITIONAL

Na página Additional encontram-se outros componentes de interface.

Página Additional: componentes especializados

BitBtn
Cria um botão que pode conter, além do texto, uma figura bitmap.

SpeedButton
Cria um botão que pode conter um texto e uma figura. Este tipo de botão é freqüentemente utilizado em grupos, em uma barra de ferramentas.

MaskEdit
Apresenta um campo de edição onde o usuário digita uma única linha de texto, similar ao componente Edit. A diferença é que o MaskEdit fornece uma forma de especificar um formato particular, tal como CEP, número de telefone, etc.

StringGrid
Cria uma tabela que pode ser usada para apresentar textos divididos em colunas e linhas.

DrawGrid
Cria uma tabela que pode ser usada para apresentar diversos tipos de dados em forma de tabela.

Image
Apresenta figuras bitmap, icon, etc. em um formulário.

Shape
Desenha figuras geométricas no formulário.

Bevel
Cria linhas e caixas tridimensionais no formulário.

ScrollBox
Cria uma área que apresenta automaticamente barras de rolagem (scrollbars) quando necessário.

CheckListBox
Apresenta um lista similar ao ListBox, exceto que cada item possui também um CheckBox.

Splitter
Um componente Splitter pode ser colocado entre dois componentes para permitir ao usuário alterar o tamanho desses componentes em tempo de execução.

StaticText
Este componente possui um texto não editável, como o componente Label. A diferença é pode-se definir uma moldura ao redor do texto.

Chart
Este componente desenha gráficos em um formulário utilizando uma pequena planilha como fonte de dados

WIN32

Os página de componente Win32 fornece a maioria dos controles encontrados na interface Windows 95/98/NT.

Página Win32: aplicativos com a aparência do Windows 95/NT.

TabControl

Fornece uma área contendo uma única página de informação e um conjunto etiquetas (Label).

PageControl

Define múltiplas paginas ou seções de informação em uma mesma janela. Possui etiquetas para acessar cada uma dessas páginas.

ImageList

Coleção de imagens do mesmo tamanho que podem ser referenciadas através de um índice. Este componente é usado para se gerenciar eficientemente um grande conjunto de figuras (ícones ou bitmaps).

RichEdit

Define uma área de texto que suporta a definição de tipos diferentes de letras (fontes), além de propriedades de formatação tais como alinhamento, tabulações, etc.

TrackBar

Define um determinado valor através do deslocamento de um apontador sobre um régua. Pode-se configurar a orientação como vertical ou horizontal.

ProgressBar

Barra retangular que é preenchida da esquerda para a direita, como aquela mostrada quando se copia arquivos no Windows Explorer. Fornece um retorno visual sobre o progresso de uma operação.

UpDown

Incrementa e decrementa valores através de uma seta para cima e uma para baixo..

HotKey

Cria e apresenta combinações de teclas que poderão ser usadas como teclas de atalho.

Animate

Apresenta um vídeo .AVI (Audio Video Interleaved). Possui alguns vídeos pré definidos como aqueles que são apresentados pelo Windows 95 quando se exclui ou copia um arquivo, por exemplo..

DateTimePicker

Apresenta um calendário para a seleção de data. O usuário pode digitar, selecionar através de um calendário ou através de setas para cima e para baixo.

TreeView

Permite apresentar informações através de uma estrutura hierárquica.

ListView

Fornece diferentes formas de apresentar informações sob forma de lista.

HeaderControl

Apresenta um cabeçalho que pode ser dividido colunas contendo textos ou números. Pode-se configurar o comportamento de cada uma dessas colunas para que, como um botão, execute uma função especifica quando o usuário clicar sobre ele.

StatusBar

Define uma área para apresentar o estado atual de alguma operação, mensagens de erro ou de alerta, avisos, etc.

ToolBar

Agrupa um conjunto de botões e outros componentes, organizando-os em colunas e ajustando automaticamente seus tamanhos e posições.

CoolBar
Apresenta uma coleção de linhas (CoolBand) que podem ser movimentados pelo usuário em tempo de execução.

SYSTEM

A página System possui controles especializados que interagem com o Sistema Operacional ou com algumas características de Hardware.

Página System: Interação com o Sistema Operacional

Timer

Timer é um componente não-visual que dispara um evento em determinada hora ou repetidamente após um determinado intervalo de tempo.

PaintBox

Apresenta uma área onde é permitido apresentar e editar figuras. Este componente é utilizado em aplicações de desenho.

MediaPlayer

Apresenta um painel de controle para apresentar e gravar informações multimídia.

OleContainer

Cria uma área **Object Linking and Embedding** (OLE) em um formulário.

DdeClientConv

DDE é uma antiga tecnologia para implementação de comunicação. Para novos projetos deve-se utilizar OLE.

O componente DDEClientConv fornece às aplicações a habilidade de estabelecer uma conversão DDE com uma aplicação servidora DDE. Este componente deve trabalhar em conjunto com um componente DDEClientItem para que sua aplicação seja uma aplicação cliente DDE completa.

DdeClientItem

O componente DDEClientItem define o item da conversão DDE. Utilize este componente junto com o componente DDEClientConv para implementar uma aplicação cliente DDE.

DdeServerConv

Este componente fornece à aplicação a habilidade de estabelecer uma conversão DDE com um aplicação cliente. Ao utilizar este componentes, sua aplicação será uma servidor DDE. Este componente é utilizado em conjunto com o componente DDEServerItem para que sua aplicação seja um servidor DDE completa.

DdeServerItem

O componente DDEServerItem define o tópico de uma conversão DDE com outra aplicação. Este componente é utilizado em conjunto com um componente DDEServerConv.

INTERNET

Os componentes da página Internet permitem implementar aplicações que utilizem algum tipo de serviço Internet.

Pagina Internet: Componentes para aplicações Internet

ClientSocket

Acrescente este componente a um formulário ou Data Module da sua aplicação para torná-la uma aplicação cliente TCP/IP.

O componente ClientSocket define uma conexão a um servidor TCP/IP, gerencia a abertura e a finalização de uma conexão.

ServerSocket

Acrescente este componente a um formulário ou Data Module para tornar sua aplicação um servidor TCP/IP.

O componente ServerSocket recebe as requisições TCP/IP de outras máquinas e estabelece tais conexões quando requisitadas.

WebDispatcher

Converte um Data Module em um WEB Module e permite que o servidor de aplicação Web responda às mensagens HTTP.

PageProducer

Converte um seqüência HTML em uma a string de comando HTML que pode ser interpretada por uma aplicação cliente, por exemplo, um Browser.

QueryTableProducer

Monta um seqüência de comandos HTML para gerar uma apresentação tabular de registros provenientes de um objeto Query.

DataSetTableProducer

Monta uma seqüência de comandos HTML para gerar uma apresentação tabular de registros provenientes de um objeto DataSet.

FTP

Controle ActiveX que fornece acesso fácil aos serviços **Internet File Transfer Protocol** (FTP) para transferência de arquivos e dados entre uma máquina remota e uma local.

HTML

Controle ActiveX que implementa um visualizador HTML, com ou sem recuperação automática de documentos HTML e fornece interpretador HTML, assim como uma visão de uma página HTML. O componente HTML pode ainda ser usado como um interpretador HTML não-visual para analisar ou processar documentos HTML.

HTTP

Controle ActiveX que implementa o Protocolo Cliente HTTP, permitindo aos usuários recuperarem documentos HTTP diretamente, sem possibilidade de navegação.

NNTP

Controle ActiveX que permite às aplicações acessarem o Protocolo **NNTP**, possibilitando leitura e escrita de mensagens no servidor de **news**

POP

Controle ActiveX que recupera correios eletrônicos de servidores UNIX ou outro servidor que suporta o protocolo POP3.

SMTP

Controle ActiveX que permitem as aplicações acessarem a servidores de correio eletrônico SMTP e a capacidade de enviar mensagens.

TCP

Controle ActiveX que implementa o protocolo TCP para aplicações cliente e aplicações servidoras, fornecendo fácil acesso aos seus serviços.

UDP

Controle ActiveX que fornece fácil acesso aos serviços do protocolo **UDP** (UDP). Implementa o WinSock para cliente e servidor, e representa um ponto de comunicação utilizando serviços UDP. Pode ainda ser usado para enviar e receber dados UDP.

DATA ACCESS

De uma forma geral, os componentes da página Data Access representam e dão acesso aos dados armazenadas fisicamente.

Página DataAcess: Componentes de acesso a Banco de Dados

DataSource

Age como um condutor entre um componente Table, Query, StoredProc e componentes de apresentação de dados tais como DBGrid, DBEdit, etc.

Table

Recupera dados de um tabela através do BDE, fornecendo-os a um ou mais componentes DataSource.

Query

Utiliza declarações SQL para recuperar dados de uma tabela através do BDE, fornecendo-os a um ou mais componentes do DataSource.

StoredProc

É utilizado quando uma aplicação cliente deve executar um procedimento armazenado no servidor de banco de dados. É constituído de um grupo de instruções armazenados como se fosse uma tabela, e executa uma tarefa que se repete com freqüente no servidor, retornando alguns resultados para a aplicação cliente.

Database

Configura uma conexão a um banco de dados, especialmente um banco de dados remoto.

Session

Fornece controles a grupos de componentes DataBase que podem ser associados a ele. Um componente Session default é automaticamente criado para cada aplicação de Banco de Dados. A criação manual de componentes Session é necessária somente na criação de aplicações multithread que utilizam de banco de dados.

BatchMove

Copia a estrutura de uma tabela ou os seus dados. Pode ser usado para copiar tabelas inteiras de um determinado formato para outro.

UpdateSQL

Permite utilizar o recurso de **cached updates** com um DataSet de apenas leitura. Por exemplo, poderia se utilizar um componente UpdateSQL para possibilitar a atualização de um DataSets abilitando a atualização de DataSets de apenas leitura.

Provider

Fornece dados de uma aplicação multi-tiered servidora para um DataSet de uma aplicação cliente (ClientDataSet).

ClientDataSet

É utilizado como se fosse um componente DataSet na parte cliente de um aplicação multi-tiered.

RemoteServer

Em uma aplicação cliente, fornece a conexão a uma servidor de dados remoto.

Report

Permite que uma aplicação utilize relatórios desenvolvidos em geradores de relatórios independentes tal como o ReportSmith, que era distribuído juntamente com a versão 2.0 do Delphi.

DATA CONTROLS

A página Data Controls fornecem componentes que permitem construir formulários para trabalhar diretamente com dados (campos) que estão armazenados em tabelas de banco de dados.

Página DataControls: Componentes de interface para Banco de Dados

DBGrid
Permite visualizar e editar dados de forma tabular, como uma planilha. Pode-se determinar as colunas que serão visualizadas, o formato, a ordenamento, etc.

DBNavigator
Conjunto de botões que permitem fazer todas as operações necessárias para a manutenção uma tabela de banco de dados.

DBText
Apresenta o valor de um campo do registro corrente de uma tabela.

DBEdit
Apresenta uma caixa de edição onde pode-se visualizar ou alterar o valor de um campo no registro corrente de uma tabela.

DBMemo
Permite visualizar e editar campos do tipo MEMO.

DBImage
Permite visualizar e editar imagens bitmap armazenadas no campo de uma tabela.

DBListBox
Permite visualizar e editar um campo cuja lista de possíveis valores provém de uma lista fixa de valores pré-definidos.

DBComboBox
Componente to tipo Combo, que permite visualizar e editar um campo de uma tabela cuja lista de possíveis valores provém de uma lista previamente definida.

DBCheckBox
Apresenta ou edita um campo booleano de uma tabela.

DBRadioGroup
Grupo de botões do tipo RadioButton que apresenta um conjunto de valores que podem ser armazenados em um campo de uma tabela.

DBLookupListBox
Apresenta um campo de uma tabela cuja lista de possíveis valores provém de um campo de outra tabela.

DBLookupComboBox
Componente do tipo Combo, que permite visualizar e editar um campo de uma tabela cuja lista de possíveis valores provém de um campo de outra tabela.

DBRichEdit
Controle de edição que permite apresentar e editar campos memo formatados.

DBCtrlGrid
Permite apresentar e editar dados de forma tabular, sendo que cada célula pode conter vários campos de um mesmo registro.

DBChart
Este componente desenha gráficos em um formulário utilizando uma tabela de banco de dados como fonte de dados..

DECISION CUBE

Os componentes da página Decision Cube (disponíveis somente na versão Client/Server) acrescenta recursos para análise multidimensional de dados através de referência cruzada entre tabelas de banco de dados, utilizando as poderosas característica da linguagem SQL.

Página DecisionCube: Análise multidimensional de dados

DecisionCube
Um depósito de dados multidimensional.

DecisionQuery
Forma especializada de Query usada para definir dados em um Decision Cube.

DecisionSource
Define o pivô atual de um componente DecisionGrid ou um componente DecisionGraph.

DecisionPivot
Botões utilizados para abrir ou fechar dimensões ou campos de um DecisionCube.

DecisionGrid
Apresenta dados multidimensionais de forma tabular.

DecisionGraph
Apresenta campos de um DecisionGrid como um gráfico dinâmico que se altera quando suas dimensões são modificadas.

QREPORT

Os componentes da página QReport permitem construir relatórios através da definição de da definição de faixas, títulos, cabeçalhos, sumários, grupos, etc. Pode-se gerar relatórios a partir de qualquer tipo de DataSource: Table, Query, etc. Durante a montagem do relatório é possível visualiza-lo através do Preview.

Página QReport: Componentes para a construção de relatórios

QuickRep

Formulário básico para a construção de relatórios, onde são colocados todos componentes do relatório. É um componente visual que possui o tamanho do papel selecionado (A4, Legal, etc.). Um relatório é criado desenhando-se faixas e outros componentes sobre um QuickRep conectado a um DataSet.

QRSubDetail

Faz a ligação de DataSets adicionais em um relatório. Tipicamente é utilizado para construir relatórios do tipo Master/Detail.

QRBand

Adicione um componente QRBand em seu relatório e configure a propriedade BandType para especificar o seu comportamento durante a geração do relatório.

QRChildBand

Faxas filhas (Child Bands) são usadas como extensões das faixas regulares de um relatório.

QRGroup

Define o inicio e o final de um grupo de registros conforme determinado atributo. É utilizado para a definição das "quebras" de um relatório.

QRLabel

Imprime um texto ou outra informação que não provém de um banco de dados.

QRDBText

É uma versão do componente QRLabel que imprime o valor de campos calculados e campos texto, incluindo campos string, numérico, data e memo.

QRExpr

Imprime campos de banco de dados, cálculos e textos estáticos.

QRSysData

Imprime informações do sistema, tais como data, hora, número da página, etc.

QRMemo

Imprime uma grande quantidade de texto que não provém de um campo de banco de dados Pode ser texto estático ou pode-se alterar durante a geração do relatório..

QRRichText

Permite imprimir no relatório um texto formatado.

QRDBRichText

Permite imprimir no relatório o texto formatado armazenado em um campo de uma tabela.

QRShape

Apresenta figuras simples tais como retângulos, círculos e linha em um relatório.

QRImage

Apresenta uma figura em um relatório.

QRDBImage

Imprime imagens armazenadas em campos de banco de dados.

QRCompositeReport

QRPreview

Define uma área para a visualização de relatórios, no formato em que iriam ser impressos.

QRChart

Permite a impressão de diversos tipos gráficos.

DIALOGS

Os componentes da página Dialogs implementam as caixas de diálogo do Windows 95/NT. Caixas de diálogo fornecem uma interface consistente para operações com arquivo, tais como abertura, gravação e impressão.

Página Dialogs: Caixas de diálogo já prontas.

OpenDialog
Apresenta uma caixa de diálogo para abertura de arquivos, comum nas aplicações Windows 95/NT.

SaveDialog
Apresenta uma caixa de diálogo para gravar arquivos, comum nas aplicações Windows 95/NT

OpenPictureDialog
Apresenta uma caixa de diálogo para abertura de arquivos gráficos, onde o usuário pode visualizar a imagem armazenada no arquivo selecionado.

SavePictureDialog
Apresenta uma caixa de diálogo para gravação de arquivos gráficos. Idêntica ao SaveDialog, exceto que possui uma área onde é apresentado a imagem que será gravada.

FontDialog
Apresenta uma caixa de diálogo que permite especificar o tipo de letra (fonte), seu tamanho e estilo.

ColorDialog
Apresenta a caixa de diálogo para seleção de cores.

PrintDialog
Mostra uma caixa de diálogo que permite especificar informações de impressão, tais como as páginas a serem impressas, o número de cópias, etc.

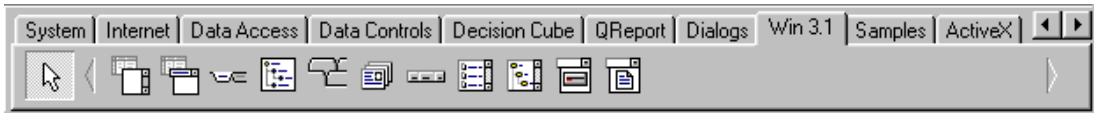
PrinterSetupDialog
Mostra uma caixa de diálogo que permite a configuração da impressora.

FindDialog
Apresenta uma caixa de diálogo que permite especificar uma String de pesquisa.

ReplaceDialog
Mostra uma caixa de diálogo que permite especificar uma String de pesquisa e uma String para substituição.

Win 3.1

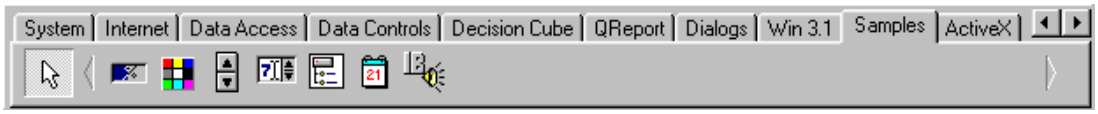
Possui componentes de interface no padrão Windows 3.1. Estes componentes servem para manter a compatibilidade com as aplicações desenvolvidas em Delphi 1.0 (Windows 3.1).



Página Win3.1: Compatibilidade com o Windows 3.1

SAMPLES

Os componentes dessa página são exemplos de componentes personalizados. O código fonte destes componentes estão disponíveis no diretório DELPHI 3\SOURCE\SAMPLES (instalação Default).



Página Samples: Alguns exemplos de componentes.

Gauge
Barra retangular que é preenchida da esquerda para a direita. Fornece um retorno visual sobre o progresso de uma operação.

ColorGrid
Apresenta uma tabela de cores.

SpinButton
Semelhante ao componente UpDown. Uma diferença é que o componente SpinButton permite definir uma figura para os botões.

SpinEdit
Semelhante ao componente UpDown. Uma diferença é que já apresenta um campo onde é apresentada a informação.

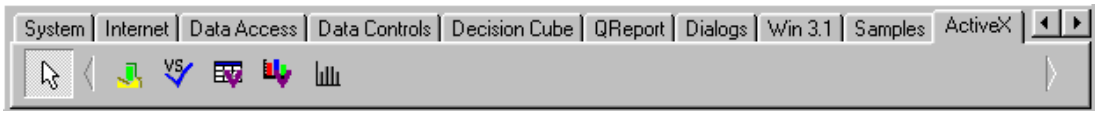
DirectoryOutline
Apresenta a árvore de diretórios do disco, semelhante ao que é apresentado pelo Windows/NT Explorer.

Calendar
Implementação de um calendário.

IBEVentAlerter

ActiveX

Os componentes na página ActiveX são objetos completos apresentados como exemplos. Para utiliza-los deve-se primeiro abrir um formulário ActiveX em um projeto ActiveX. Após posicionar um componente em um formulário ActiveX, clique o botão direito do mouse para ser apresentadas as propriedades e outros comandos.



Página ActiveX: Exemplos da utilização de componentes ActiveX

Chartfx
Permite criar gráficos personalizados. Selecione as propriedades para apresentar um painel de controle que permite definir valores, aparência e algumas outras características.

VSSpell
Visual Speller, um corretor ortográfico personalizado.

F1Book
Formula One. Permite implementar uma planilha de cálculo.

VtChart
Permite-se criar gráficos 3D.

VtChart
Pinnacle Graph, permite gerar gráficos 2D.

5

Projetos Delphi

Os Arquivos do Projeto

Um projeto Delphi é composto por vários arquivos relacionados entre si. Alguns destes arquivos são gerados durante o desenvolvimentos, quando está se definindo os formulários da aplicação. Outros arquivos são gerados na compilação da aplicação.

O Arquivo Principal (.DPR)

O arquivo principal é gerado durante a construção da aplicação e possui a extensão **.dpr**. É neste arquivo que é instanciado o formulário principal da aplicação, assim como todos os formulários que serão automaticamente instanciados pelo Delphi. Você raramente irá editar o arquivo de projeto de uma aplicação

```
program Project1;

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Note que neste arquivo existe uma cláusula **uses** contendo a relação de uma **Unit**. O arquivo de projeto irá listar todas as **Units** que compõem a aplicação, estando elas relacionadas a formulários ou não. Outro ponto que vale ressaltar é a diretiva de compilação **{$R *.RES}**. As chaves seguidas pelo caracter **$** indicam uma diretiva de compilação, ou seja, um determinado comportamento do compilador Delphi. Neste caso, esta diretiva indica que o compilador irá adicionar os arquivos com extensão **.RES** (arquivos de recurso), quando da link-edição do projeto.

O Arquivo de Recurso (.RES)

O arquivo de recurso é um arquivo binário gerado automaticamente pelo Delphi que contém basicamente o ícone da aplicação. Poderá ser adicionado um título a sua aplicação, ou gerar um outro arquivo de recursos para o seu projeto. Por *default*, todo o projeto Delphi possui um arquivo de recurso que terá o mesmo nome do arquivo principal (**.dpr**) e será "linkado" ao arquivo executável (**.exe**).

O Arquivo de Opções (.DOF)

Um arquivo de opções de projeto será gerado quando sua aplicação é salva pela primeira vez. Este arquivo irá conter opções de compilação de projeto, indicações de criação automática de formulários, etc. Pode-se alterar as opções de um projeto utilizando o comando **Option** do menu **Project**.

Além do arquivo **.dof**, um arquivo com extensão **.DSK** será criado com as opções do ambiente de desenvolvimento Delphi. Assim como as opções de um projeto (***Project***|***Options***), pode-se definir diferentes opções de ambiente para cada projeto. Para se alterar as opções de ambiente utiliza-se o comando **Environment Options** do menu **Tools**.

Os Arquivo de Código (.PAS)

Os arquivos **.pas** são os programas fonte em Object Pascal, as **Units**. Pode-se dividir as Units em três classes:

- units que possuem um formulário relacionado a ela;
- units relacionadas a componentes;
- units de uso geral, que possuem prodecures e functions que serão utilizadas por outros programas

```
unit Unit1;

interface

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

type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;

implementation

{$R *.DFM}

end.
```

Arquivos de Formulários (.DFM)

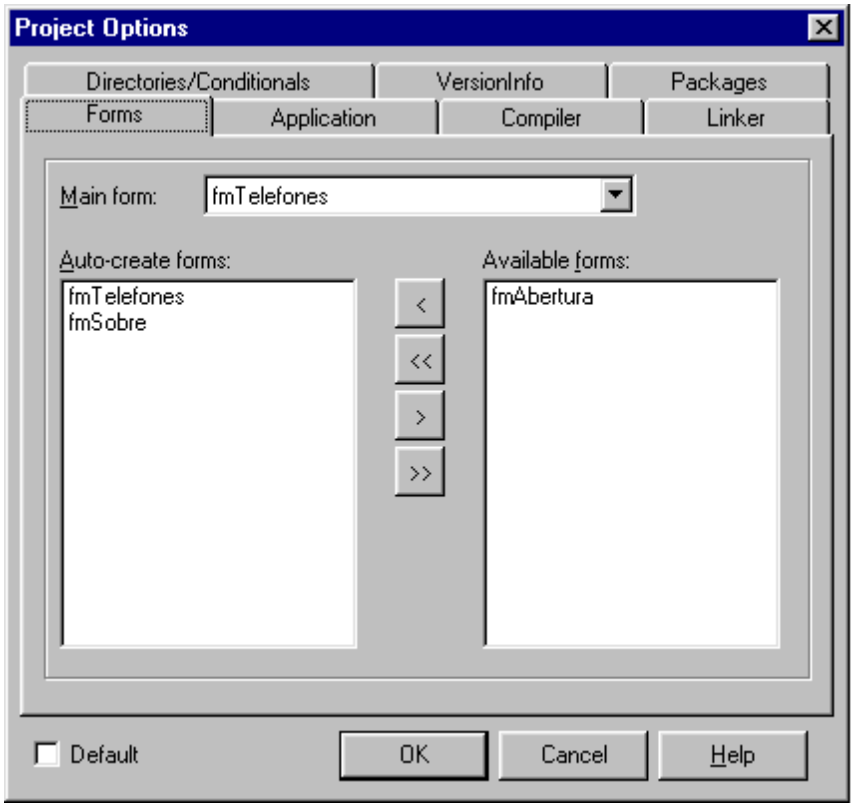
Quando um novo formulário é adicionado em uma aplicação, o Delphi gera dois arquivos correspondente ao formulário. Um arquivo é o código fonte em Object Pascal (**.PAS**). O outro arquivo (**.DFM**), descreve as características do formulário como, por exemplo, a altura, a largura, o título, a cor, etc. Raramente será necessário visualizar o conteúdo deste tipo de arquivo. Ele é atualizado automaticamente toda vez que é feito uma alteração no formulário.

Arquivos Compilados

Após a compilação de um projeto, serão gerados mais dois tipos de arquivos. Os arquivos com extensão **.DCU** (*Delphi Compiled Unit*) e o arquivo com extensão **.EXE**. Um arquivo **.dcu** é o resultado da compilação dos arquivos **.pas** e **.dfm**. O arquivo executável **.exe** é o resultado da compilação de todos os arquivos do projeto.

As Opções de um Projeto

O comando de menu Project | Options, irá apresentar uma caixa de diálogo contendo algumas opções relativas ao seu projeto. Você poderá definir desde o formulário principal da aplicação, até a forma de como o projeto será compilado.



A Página Forms

Na página Forms é possível selecionar o formulário principal da aplicação e para definir quais os formulários serão criados automaticamente pelo Delphi.

- **Main form**: Indica qual o formulário principal da sua aplicação. Este será o primeiro formulário apresentado quando se executa a aplicação. Através do ComboBox é possível alterar a definição do formulário principal da aplicação. Normalmente o formulário principal é gerado automaticamente. Isto é, ele está presente na lista **Auto-create forms**.
- **Auto-create forms**: Contém a lista dos formulários gerados automaticamente pelo Delphi quando sua aplicação for executada.
- **Available forms**: Contém a lista dos formulários que serão gerados pela aplicação. Para fazer uso dos formulários que estão sendo mostrados nesta lista eles devem ser instanciados anteriormente.

A Página Application

Nesta página poderá definir-se um título e um ícone para a aplicação.

- **Title**: Especificação de um título para a sua aplicação. Este título será apresentado quando sua aplicação estiver minimizada.
- **Help file**: Associação de um arquivo de help (**.hlp**) que será utilizado na sua aplicação. O botão Browse irá abrir uma caixa de diálogo para que você localize o arquivo de help.
- **Icon**: Especificação de um ícone para a aplicação.

A Página Compiler

Define as opções de compilação de seu projeto. Entre as opções disponíveis, pode-se indicar se serão realizadas checagem de limite de variáveis, se serão compiladas informações para depuração do projeto, etc. Todas as opções desta página são diretivas de compilação que também poderão ser colocadas diretamente no código da unit.

A Página Linker

Contém as indicações de como os arquivos do projeto serão link-editados. Nesta página indica-se se o compilador Delphi irá gerar um arquivo externo com mapeamento dos arquivos linkados, indicará também se o produto final da link-edição será uma unit compilada (**.dcu**) ou um arquivo objeto (**.obj**), e mais diversas outras opções.

A Página Directories/Conditionals

Utiliza-se esta página para definir onde os seus arquivos compilados serão armazenados e para definir um símbolo de compilação. Pode-se decidir, de acordo com o símbolo de compilação, qual a parte do código será compilada.

A Página Version Info

Esta página permite especificar informações de versão de seu projeto.

A Página Packages

"Packages" são arquivos que contém várias classes pré-definidas pelo Delphi, como, por exemplo, botões, forms, tables, etc. O nome package é uma nova designação para VCL – Visual Component Library.

Você poderá indicar quais os packages que estarão disponíveis em tempo de desenvolvimento e quais os que serão requisitados em tempo de execução. Com o controle dos packages, consegue-se reduzir o tamanho do arquivo executável, pois só serão compilados os packages que fazem parte da sua aplicação.

Universidade Federal da Paraíba
Centro de Ciências Sociais Aplicadas
Departamento de Ciência da Informação
Programa de Pós-Graduação em Ciência da Informação
Ação para cidadania e acesso livre à informação

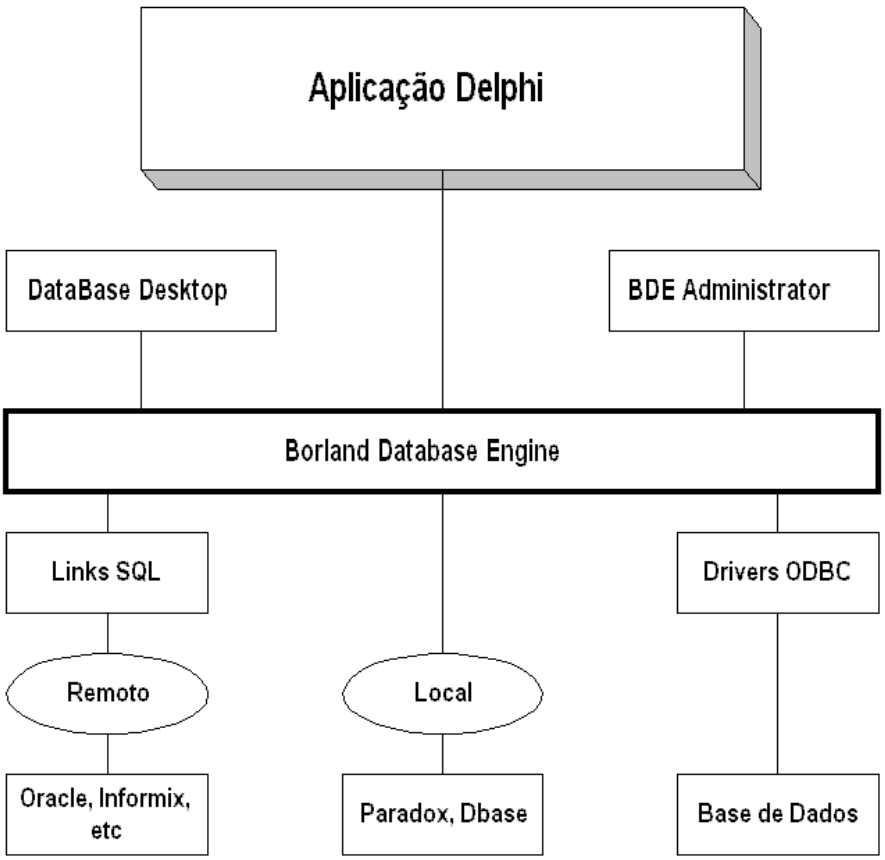
Banco de Dados

João Pessoa, Maio de 2011

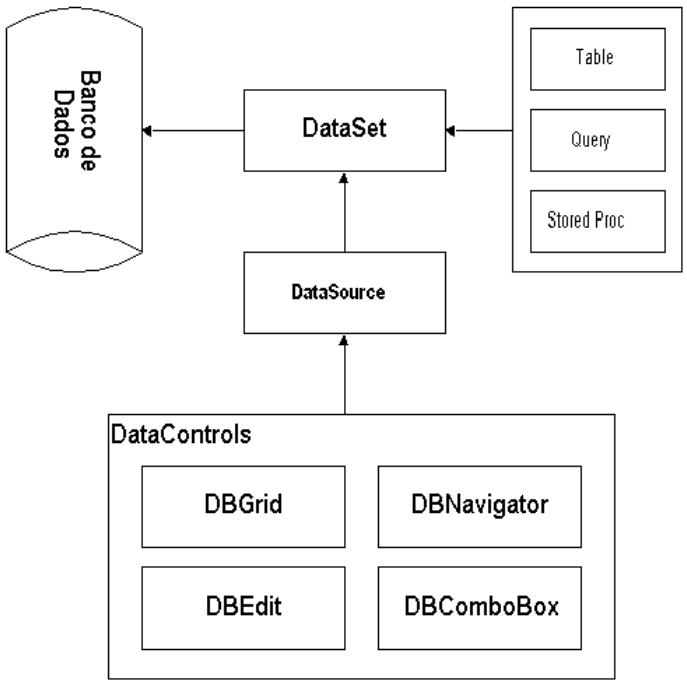
6

Delphi com Banco de Dados

O Delphi provê um ambiente bastante simples para o desenvolvimento de aplicações que acessam banco de dados. O Borland DataBase Engine (BDE) permite o acesso aos principais sistema de gerenciamento de banco de dados e de um conjunto de componentes para conexão e manipulação das bases. Com isso é possível desenvolver aplicações de forma simples e rápida.



O BDE é um conjunto de funções que possui as informações sobre os drivers de acesso dos principais gerenciadores de banco de dados do mercado. Na versão Client/Server do Delphi, o BDE traz drivers de conexão nativa para os bancos Oracle, Informix, SQL Server, DB2, entre outros.



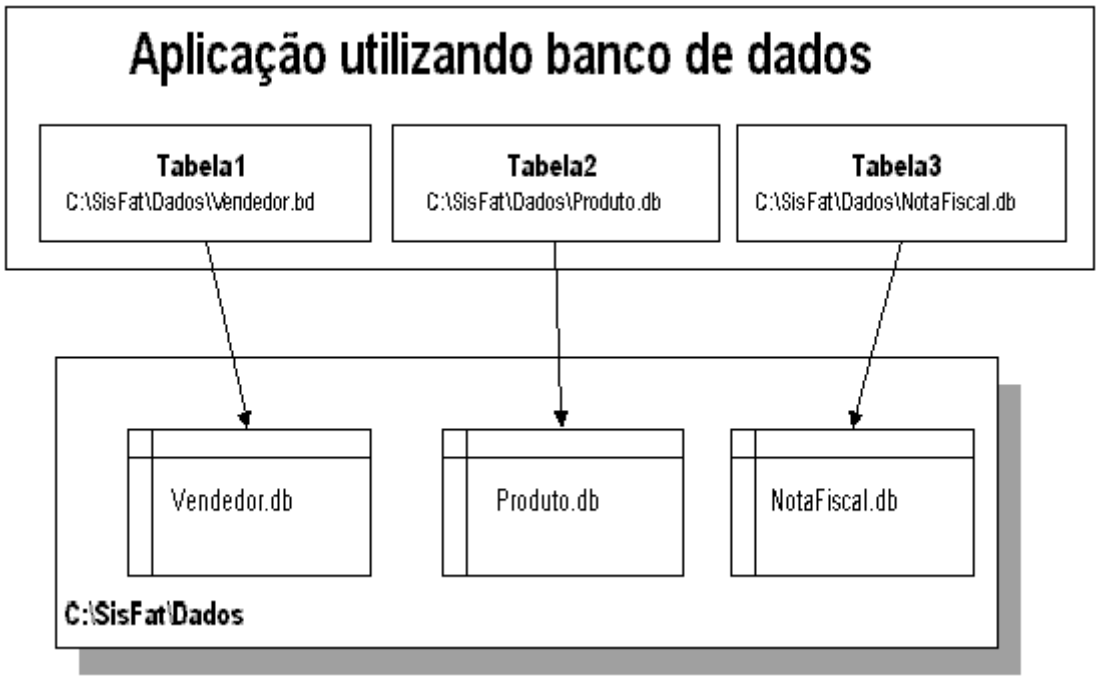
As ferramentas de gerenciamento de banco de dados do Delphi são:

- **BDE Administrator**: Configura o BDE e os drivers de Gerenciadores de Banco de Dados. Cria e configura Alias.
- **DataBase Desktop (DBD)**: ferramenta para criar, alterar e editar bancos de dados;
- **Componentes** (visuais e não visuais): permitem conexões e interface com banco de dados. Estão disponibilizados nas paletas **Data Access** e **Data Controls**;
- **Data Module**: centraliza componentes de acesso a dados dos formulários e aplicações;
- **Form Expert**: permite a geração automática de formulários-padrão para acesso a banco de dados.
- ***Object Repository***: armazena formulários e módulos de dados para serem compartilhados entre diferentes aplicações;
- ***Borland Database Engine (BDE)***: faz o relacionamento entre o DELPHI e o banco de dados;
- ***SQL Explorer***: ferramenta para gerenciar BDE e criar dicionário de dados;

- **Quick Report**: permite a geração de relatórios a partir de bancos de dados;

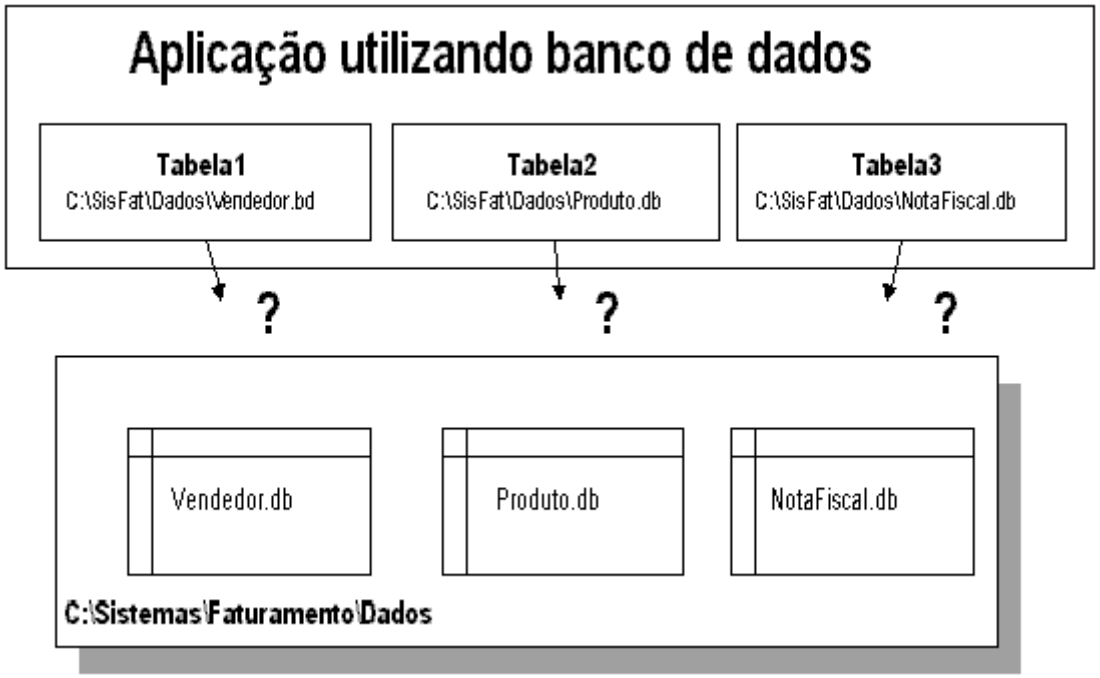
O que é um Alias ?

As aplicações que trabalham com tabelas de banco de dados podem referenciar estas tabelas utilizando o caminho (path) onde elas estão posicionadas no disco. Para cada tabela representada na aplicação é definido o caminho e o arquivo que contém os dados da tabela.



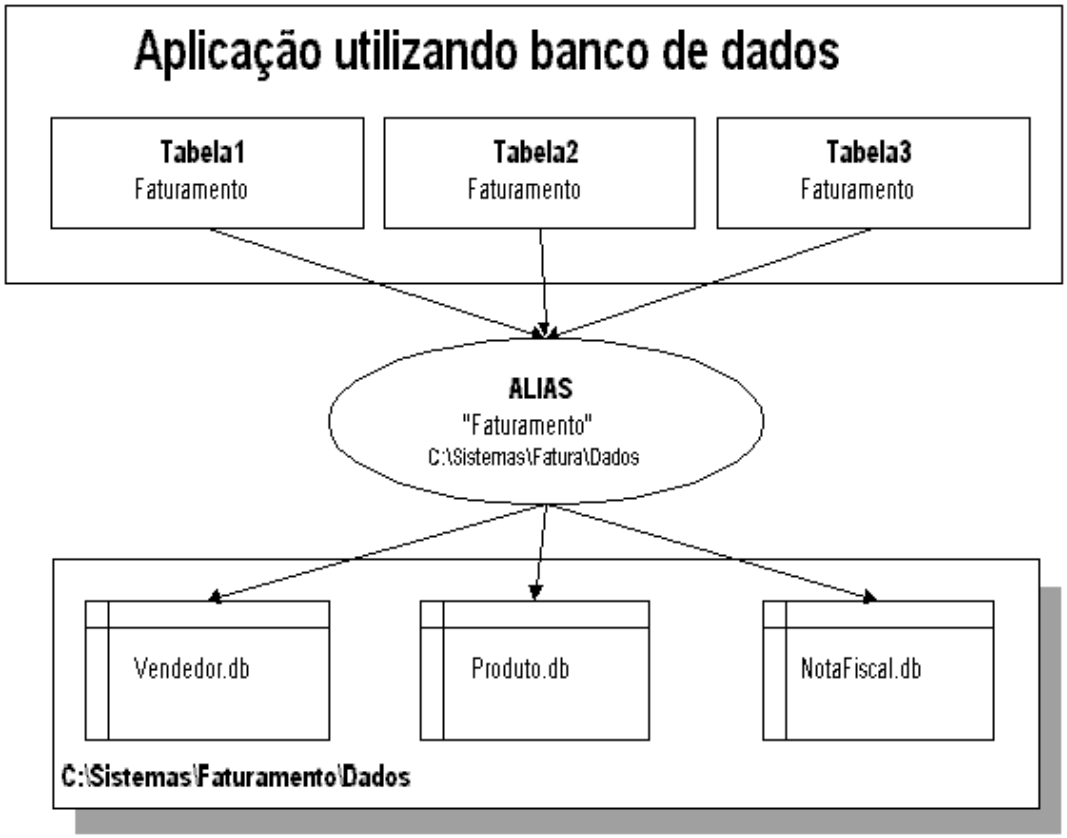
Implementação sem utilizar Alias: Referência direta ao arquivo

Apesar de funcionar sem problemas, essa forma de implementação faz com que a aplicação fique dependente da pasta (diretório) onde os dados estão localizados. Se os dados não estiverem exatamente na pasta definida na aplicação, esta não funcionará.



Alteração da pasta de dados: A aplicação não funciona mais

Uma forma de se resolver esse problema é através da utilização de um **Alias**. Um Alias referencia a pasta onde estão localizados os arquivos de dados (tabelas). A representação das tabelas representadas na aplicação referenciam o Alias previamente definido.



Desta forma, caso seja alterada a pasta onde estão localizados os dados da aplicação basta alterar a definição do Alias para que a aplicação continue funcionando normalmente.

BDE Administrator

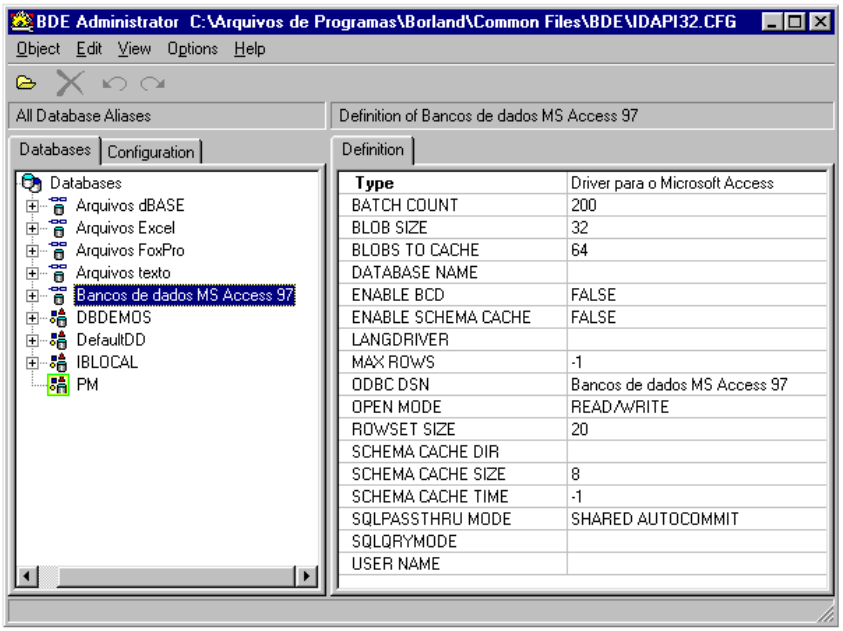
Para que se consiga acessar bases de dados a partir do Delphi, faz-se necessário a criação de referências. Estas referências são conhecidas por Alias, e são criadas em um aplicativo chamado **BDE Administrator**.

O BDE Administrator permite:

- Configurar o Borland Database Engine (BDE).
- Configurar os drivers STANDARD (Paradox e dBASE/FoxPro), os drivers SQL, Access e também criar e configurar drivers ODBC.
- Criar e manter Alias de banco de dados.

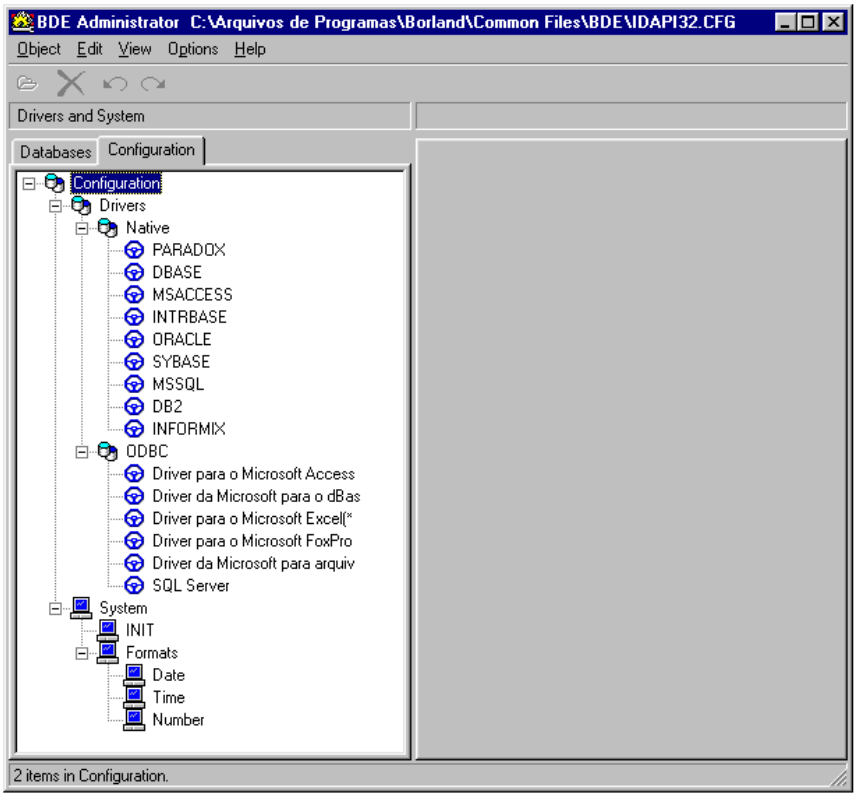
A Página Databases

Mostra os alias dos bancos de dados disponíveis.



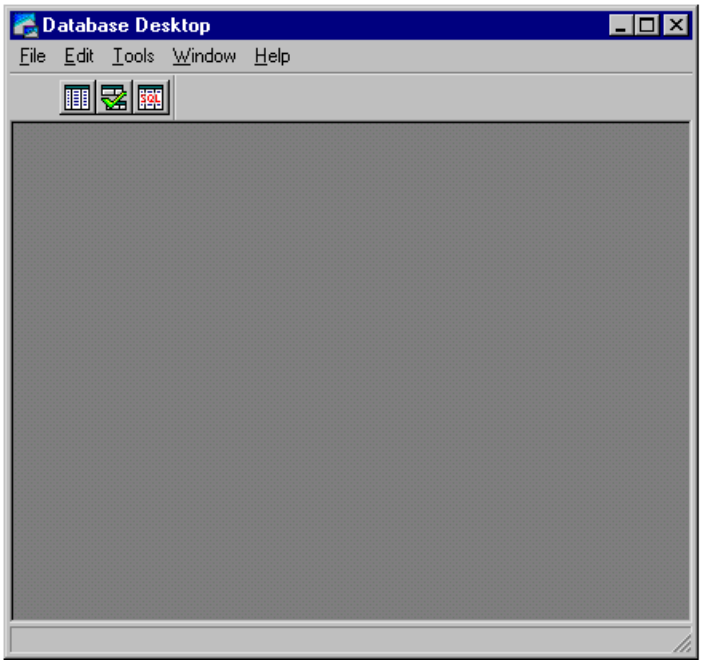
A Página Configuration

Permite adicionar e configurar drivers ODBC, configurar drivers padrão e drivers SQL e ainda configurar o Borland Database Engine (BDE).



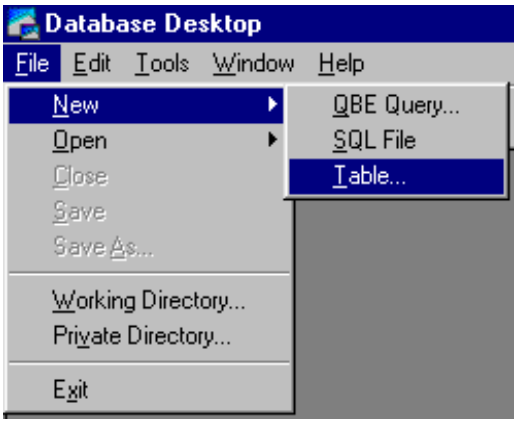
DataBase Desktop

O ***DataBase Desktop*** é uma ferramenta do DELPHI para manipulação de Banco de Dados. Ela é útil principalmente para a criação, visualização, modificação e consulta de tabelas nos formatos Paradox, dBASE e SQL.



Criando uma Tabela

Para criar uma nova tabela utiliza-se a opção do menu ***File|New|Table***.



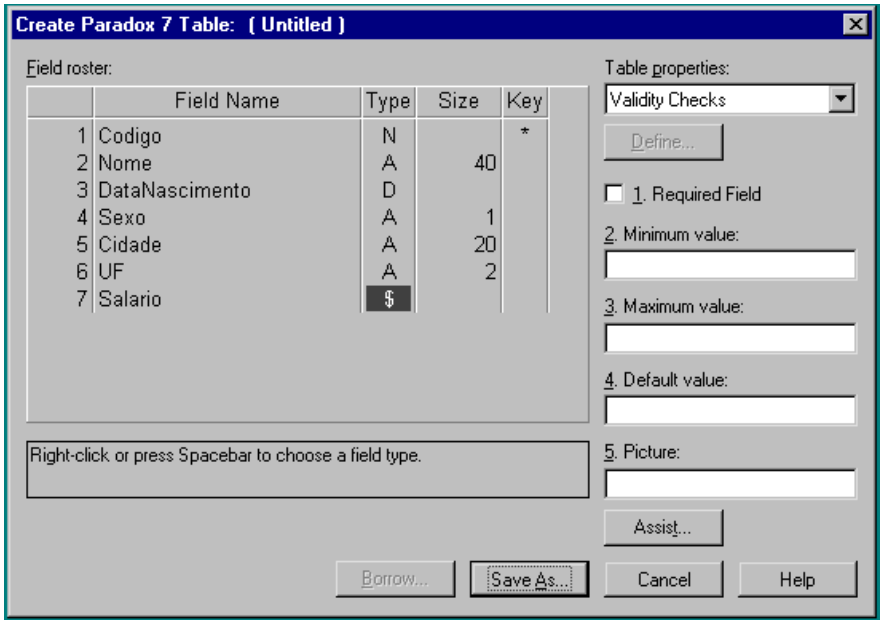
Será apresentado uma janela para a definição do tipo de tabela que se deseja criar. Clique o botão OK. Neste instante devem ser definidos os atributos dos campos, como segue:

- ***Field Name***: nome do campo;
- ***Type***: tipo do campo. Clicando o botão direito do mouse pode-se selecionar o tipo do campo;
- ***Size***: define o tamanho do campo. Só disponível para alguns tipos;

- ***Key***: define campos-chave para a tabela.

Ainda nesta tela pode-se definir:

- ***Table properties***: define características gerais dos campos da tabela;
- ***Required Field***: define se o campo tem um valor requerido;
- ***Minimum Value***: valor mínimo para o campo;
- ***Maximum Value***: valor máximo para o campo;
- ***Default Value***: valor padrão para o campo;
- ***Picture***: define o formato de entrada do campo.

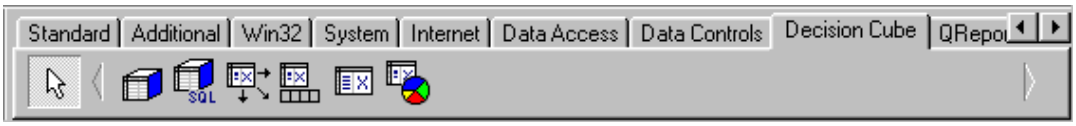


Componentes de Banco de Dados

A página **Data Access** possui componentes que permitem a conexão do DELPHI com bancos de dados.

A página **Data Controls** contém componentes que permitem a manipulação e visualização dos dados de um banco de dados.

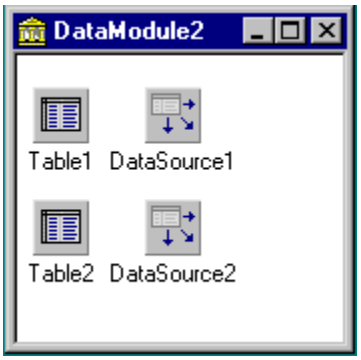
Os componentes da página **Decision Cube** fornece recursos para análise multidimensional de dados através de referência cruzada entre tabelas de banco de dados utilizando a linguagem SQL.



Data Module

O ***Data Module*** é um formulário especial que permite centralizar os componentes de controle de banco de dados. Suas principais vantagens são:

- permite que toda a aplicação utilize os mesmos componentes de controle de dados, evitando duplicação de componentes;
- garantem o uso correto das tabelas, pois elas não precisam ser recriadas em cada formulário;
- permitem definir novos eventos ou métodos sobre as tabelas que sejam comuns em toda a aplicação.



Data Module: Centralização de componentes de banco de dados.

Formulários Utilizando Banco de Dados

A criação de um formulário para a manipulação do banco de dados pode ser feita de forma manual (adicionando-se cada componente individualmente) ou de forma automática (através do **Form Expert**).

O ***Form Expert*** é uma ferramenta do DELPHI que permite criar um formulário padrão para a manipulação de banco de dados. Sua principal vantagem está na automatização do processo de criação de formulário, o que permite ganhar tempo no desenvolvimento da aplicação.

Universidade Federal da Paraíba
Centro de Ciências Sociais Aplicadas
Departamento de Ciência da Informação
Programa de Pós-Graduação em Ciência da Informação
Ação para cidadania e acesso livre à informação

TUTORIAL

João Pessoa, Maio de 2011

Universidade Federal da Paraíba
Centro de Ciências Sociais Aplicadas
Departamento de Ciência da Informação
Programa de Pós-Graduação em Ciência da Informação
Ação para cidadania e acesso livre à informação

Construindo uma Aplicação DELPHI

Serão vistas aqui as etapas para construção de uma aplicação Delphi. Como exemplo, será utilizado um sistema comercial simplificado, que utiliza tabelas de banco de dados. A seqüência com que essas etapas são apresentadas serve apenas para a condução didática na elaboração do exemplo proposto, não tem pretensão de metodologia. Pelo contrário. A forma com que serão implementadas algumas características do nosso sistema-exemplo se justifica apenas por uma questão de gosto pessoal.

João Pessoa, Maio de 2011

1

Definindo o Sistema

É sempre bom lembrar que, antes de sentar-se em frente ao micro e sair programando, análise de sistema e a modelagem de dados ainda são fundamentais para o sucesso de um projeto de sistema. É muito difícil para um programador compulsivo ficar "parado" elaborando diagramas, "perdendo tempo" rabiscando coisas que já estão claras **em sua cabeça**. Esse pensamento é o caminho mais curto para aqueles "sistemas não acabam nunca".

Sofisticados recursos de interface não conseguem ajeitar um sistema mal elaborado.

Na elaboração de sistemas que envolvam Banco de Dados não se deve esquecer de toda a teoria que envolve esta área. Modelagem de dados, normalização, etc., foram e estão sendo estudadas para serem usadas. Não se deve elaborar novas teorias ou "um **jeitinho** todo seu de se fazer as coisas". Pelo menos não durante o desenvolvimento de um sistema importante.

Toda a "papelada" gerada durante a análise do sistema, se bem padronizada, pode servir como documentação do sistema, e ainda ser um ponto de partida para a elaboração dos manuais. Dessa forma o seu sistema continuará funcionando caso o analista ou o programador saia de férias ou arranje outro emprego.

O Sistema

O sistema que será utilizado como exemplo serve para emissão de notas fiscais. Ele terá o nome de **SisFat**, e terá as seguintes atribuições:

- Controle de Produtos;

- Controle de Vendedores;
- Elaboração e emissão de Notas Fiscal.

As Tabelas

As informações estão divididas em quatro tabelas de banco de dados como definidas abaixo:

PRODUTO	VENDEDOR	NotaFiscal	ItemNota
Código	Código	Número	Nota
Descrição	Nome	Data	Item
Quantidade em estoque	Data de admissão	Vendedor	Produto
Preço de compra			Quantidade
Preço de venda			

2

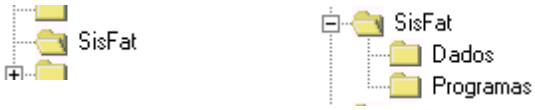
Organizando os Arquivos

Definido o sistema, iremos agora ao computador começar a programar!!! A primeira coisa a fazer é definir uma nova pasta (diretório) onde serão gravados os dados e os programas do novo sistema.

Um sistema pode conter centenas de arquivos referentes aos programas e outras centenas referentes aos dados. Portanto, é uma boa prática criar uma pasta de trabalho para a organização dessa grande quantidade de arquivos.

Criando Diretórios

Crie uma Pasta (diretório) com o nome do sistema (SisFat).



É importante separar os arquivos que se referem aos dados (tabelas, índices, etc.) dos diversos tipos de arquivos referentes aos programas (.dpr, .pas, .dfm, etc.). Para isso, na pasta **SisFat**, crie duas outras pastas: **Dados** e **Programas**.

Criando um Alias

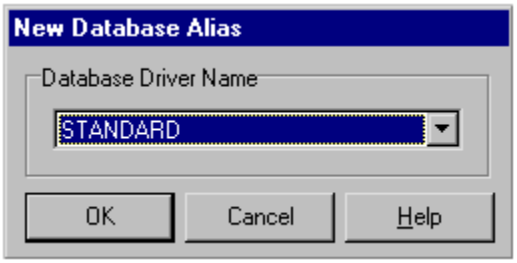
Para a aplicação proposta, vamos criar um Alias chamado **Faturamento** que referencia a pasta de dados onde serão gravadas as tabelas que iremos criar.

❑ Passo 1

Na página **Databases** do **BDE Administrator** selecione a opção **New** no menu **Object**.

❑ Passo 2

Será apresentada uma janela para a seleção do driver de banco de dados. Selecione STANDARD pois iremos trabalhar com tabelas Paradox.



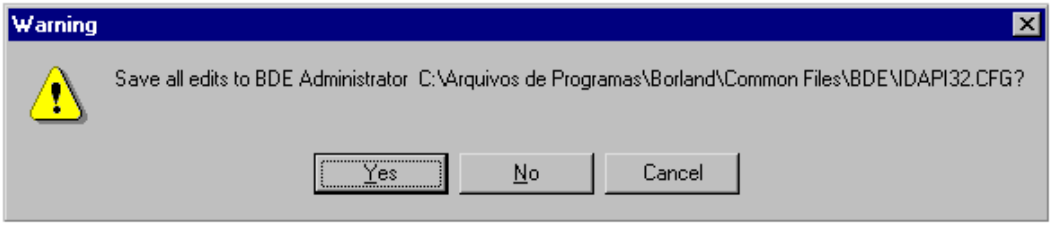
Após digitar OK, será criado um novo Alias com o nome STARDARD1. Renomeie o Alias com o nome **Faturamento**.

❑ Passo 3

Em **Definition** (lado direito da janela) preencha o atributo **Path** com o caminho onde serão posicionadas as tabelas de dados da aplicação: **C:\SisFat\Dados**. Se preferir, clique o mouse nos três pontos que aparecem no atributo para utilizar uma caixa de diálogo para auxiliar na procura do diretório de dados.

- **Passo 4**

Feche o DBE Administrator



Responda **Yes** para salvar o Alias. Como pode ser visto, todos os Alias são gravados no arquivo

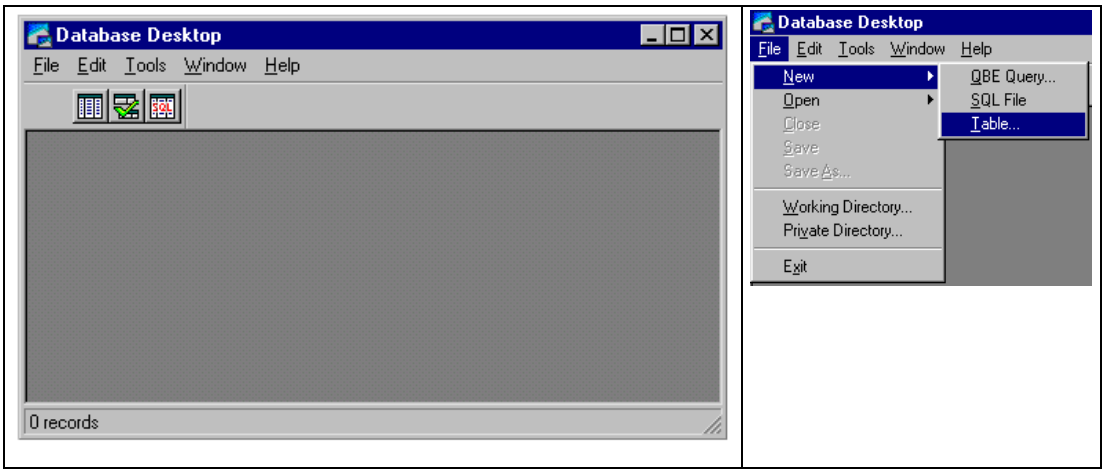
C:\Arquivos de Programas\Borland\Common Files\BDE\IDAPI32.CFG

3

Criando Tabelas

DataBase Desktop

O ***DataBase Desktop*** é uma ferramenta para manipulação de Banco de Dados. Ela é útil principalmente para a criação, visualização, modificação e consulta de tabelas nos formatos Paradox, dBASE e SQL.



No menu principal do *Database Desktop* selecione a opção ***File|New|Table***.

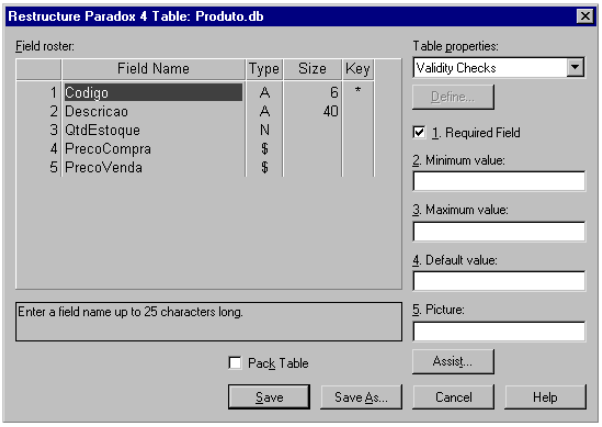
❑ PASSO 1

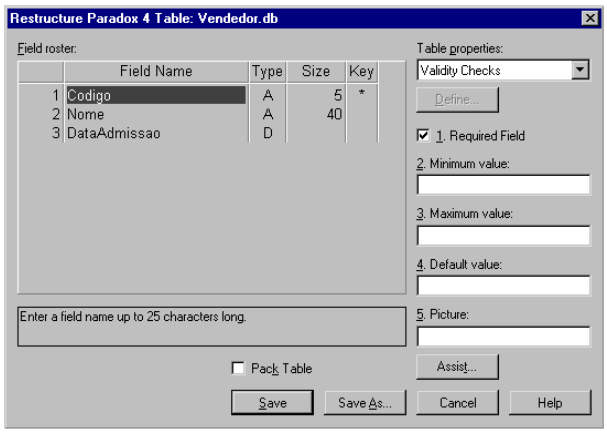
Selecione a opção ***Paradox 7*** e clique em ***OK***. As tabelas *Paradox* (extensão .DB) oferecem mais recursos que as tabelas *DBase* (extensão .DBF).

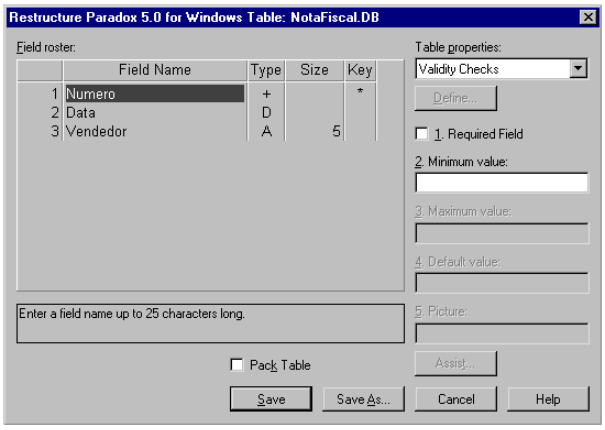
❑ PASSO 2

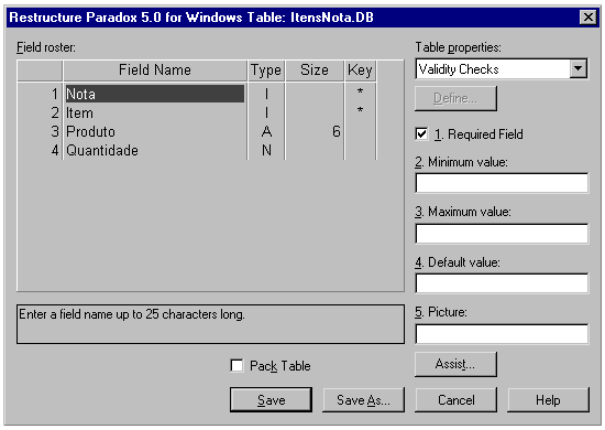
Neste instante devem ser definidos os atributos dos campos:

- ***Field Name***: nome do campo;
- ***Type***: tipo do campo. Clicando o botão direito do mouse pode-se selecionar o tipo do campo;
- ***Size***: define o tamanho do campo. Só disponível para alguns tipos;
- ***Key***: define campos-chave para a tabela.









Produto.db	**Type**	**Size**	**Key**
Codigo	A	6	*
Descricao	A	40	
QtdEstoque	N		
PrecoCompra	$		
PrecoVenda	$		

Vendedor.db	**Type**	**Size**	**Key**
Codigo	A	5	*
Nome	A	40	
DataAdmissao	D		

NotaFiscal.db	**Type**	**Size**	**Key**
Numero	+		*
Data	D		
Vendedor	A	5	

ItemNota.db	**Type**	**Size**	**Key**
Nota	I		*
Item	I		*
Produto	A	6	
Quantidade	N		

❑ PASSO 3

Clicando no botão ***Save As*** aparecerá uma caixa de diálogo para que se possa salvar a tabela em disco. Dê um nome à tabela e clique no botão ***Salvar***.

Atenção

No nosso exemplo os arquivos de dados (tabelas) devem ser gravados na pasta **C:\SisFat\Dados**.

4

Iniciando um Projeto

Ao iniciar o Delphi, é apresentado um projeto chamado **Project1** já com um formulário chamado **Form1**.

Para começarmos a partir do zero, feche (sem salvar) o formulário utilizando o comando do menu ***File|Close***. Desta forma teremos apenas o projeto sem nenhum formulário. Agora salve o projeto através do comando ***File|Save Project As***. Um arquivo de projeto possui a extensão **.dpr** (**D**elphi **Pr**oject).

Atenção

Verifique com atenção a pasta onde será gravado o projeto. No nosso exemplo os arquivos que não se referem às tabelas devem ser gravados na pasta **C:\SisFat\Programas**.

Criando um Data Module

O ***Data Module*** é um formulário especial que permite centralizar os componentes de controle de banco de dados. Suas principais vantagens são:

- permite que toda a aplicação utilize os mesmos componentes de controle de dados, evitando duplicação de componentes;
- garante o uso correto das tabelas, pois elas não precisam ser recriadas em cada formulário;
- permite definir novos eventos ou métodos sobre as tabelas que sejam comuns em toda a aplicação.

❑ PASSO 1

Selecione o comando do menu ***File|New Data Module***. Será criada uma janela chamada inicialmente DataModule1. Na propriedade **Name** digite **dmFaturamento**.

❑ PASSO 2

Inicialmente ponha na janela (Data Module) um componente **Database**, localizado no conjunto de componentes **DataAccess** da Paleta de Componentes

Altere as seguintes propriedades:

Properties	
AliasName	Faturamento
DatabaseName	dbFaturamento
Name	dbFaturamento
Connected	True

❑ PASSO 3

Ponha um componente **Table** na janela. O componente **Table** também está localizado na página **DataAccess** da Paleta de Componentes

Altere as seguintes propriedades:

Properties	
DatabaseName	DbFaturamento
TableName	Produto
Name	TbPRODUTO
Active	True

❑ PASSO 4

Acrescente um componente **DataSource** no Data Module.

Altere as seguintes propriedades:

Properties	
DataSet	TbPRODUTOS
Name	DsPRODUTO

❑ PASSO 5

Conforme os passos 3 e 4, acrescente mais três componentes **Table** e mais três componentes **DataSource** com as seguinte as propriedades:

Properties	
DatabaseName	dbFaturamento
TableName	Vendedor
Name	tbVENDEDOR
Active	True

Properties	
DataSet	tbVENDEDOR
Name	dsVENDEDOR

Properties	
DatabaseName	dbFaturamento
TableName	NotaFiscal
Name	tbNotaFiscal
Active	True

Properties	
DataSet	tbNotaFiscal
Name	dsNotaFiscal

Properties	
DatabaseName	dbFaturamento
TableName	ItensNota
Name	tbItensNota
Active	True

Properties	
DataSet	tbItensNota
Name	dsItensNota

❑ PASSO 6

Salve o Data Module utilizando o comando ***File|Save As***. O nome do arquivo será **DadosFaturamento**.

Atenção

Este arquivo deve ser gravados na pasta **C:\SisFat\Programas**.

5

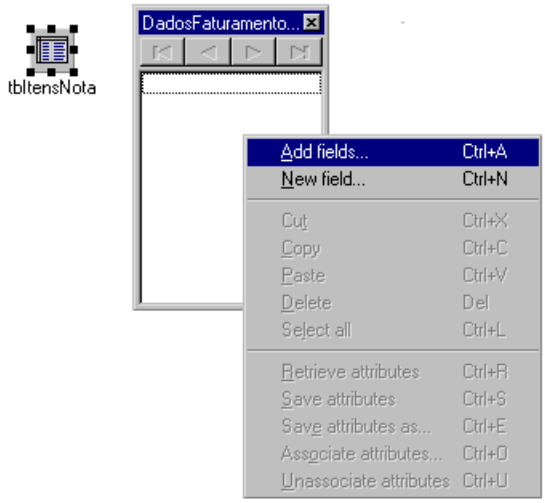
Transformando campos em objetos

Pode-se dizer que a função de um componente **Table** é representar internamente à aplicação uma tabela gravada no disco, permitindo que a aplicação consiga referenciá-la.

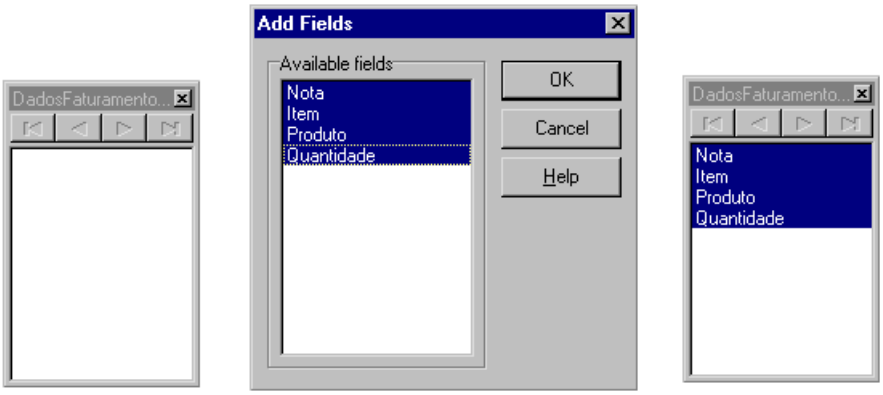
Na maioria das aplicações é necessário que cada campo de uma tabela seja também representado por um objeto. Dessa forma é possível manipular algumas propriedades de cada campo da tabela individualmente.

Como exemplo será utilizada a tabela **tbItensNota**

Dê um duplo-clique sobre a tabela **tbItensNota**. Aparecera uma pequena janela inicialmente vazia. Sobre esta janela clique o botão direito do mouse e escolha a opção ***Add Fields...***.



Será apresentada uma outra janela contendo todos os campos da tabela. Marque todos os campos e clique o botão ***OK***.



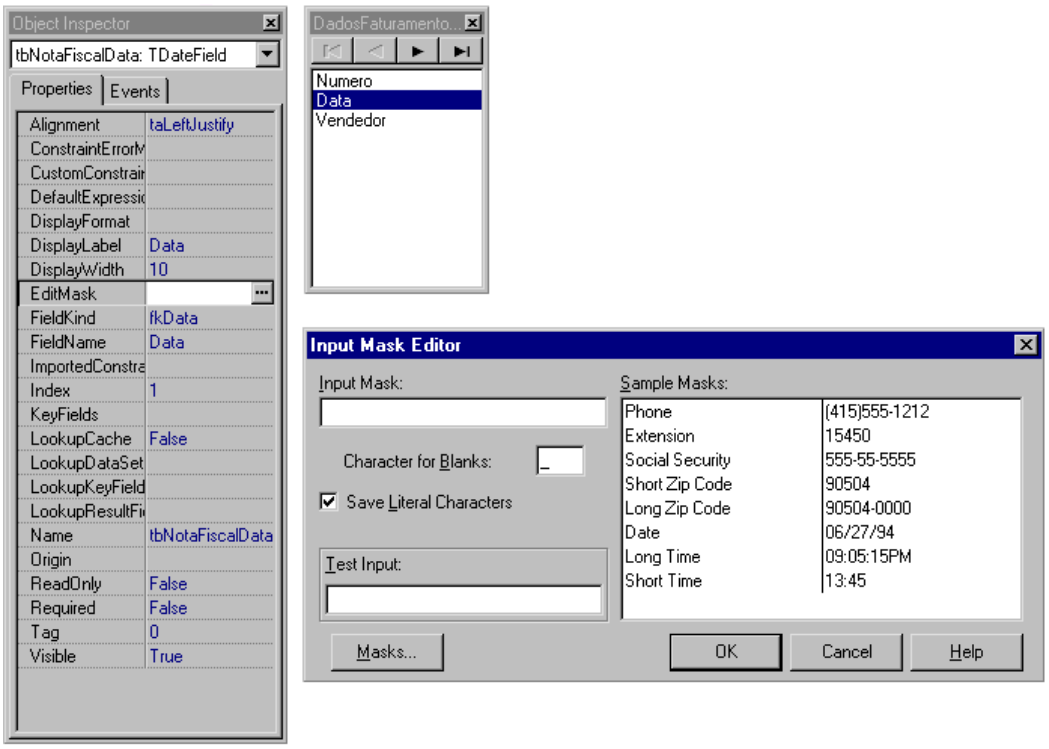
O nome de cada objeto-campo será definido automaticamente pelo Delphi. Ele será composto do nome da tabela seguido do nome do campo. Por exempo, o nome do objeto-campo definido para o campo ***Produto*** da tabela ***tbItemNota*** será **tbItemNotaProduto**.

Alterando propriedades dos campos

Após a transfomação dos campos em objetos, cada campo-objeto terá suas propriedades e eventos individualmente.

Sempre que for necessário ter acesso aos objetos-campo de uma tabela basta dar um duplo-clique sobre a tabela.

Uma das propriedades que normalmente se altera em um objeto-campo é EditMask, que define a máscara de edição de um campo.



6

Criando campos “virtuais”

O recurso de transformar campos em objetos permite que se crie objetos-campo que não estão ligados fisicamente a um campo da tabela.

Na tabela **tbItemNota**, por exemplo, temos o código do **Produto** e a **Quantidade**. Sabemos antecipadamente que durante a execução do nosso sistema teremos que calcular diversas vezes o valor total do item (preço de venda * quantidade). Porém o preço de venda do produto não aparece nesta tabela e sim na tabela **tbPRODUTO**. *“Que bom seria se o campo **PrecoVenda** aparecesse também na tabela **tbItemNota**, economizaria muito trabalho!!!”.*

Criando campos *lookup*

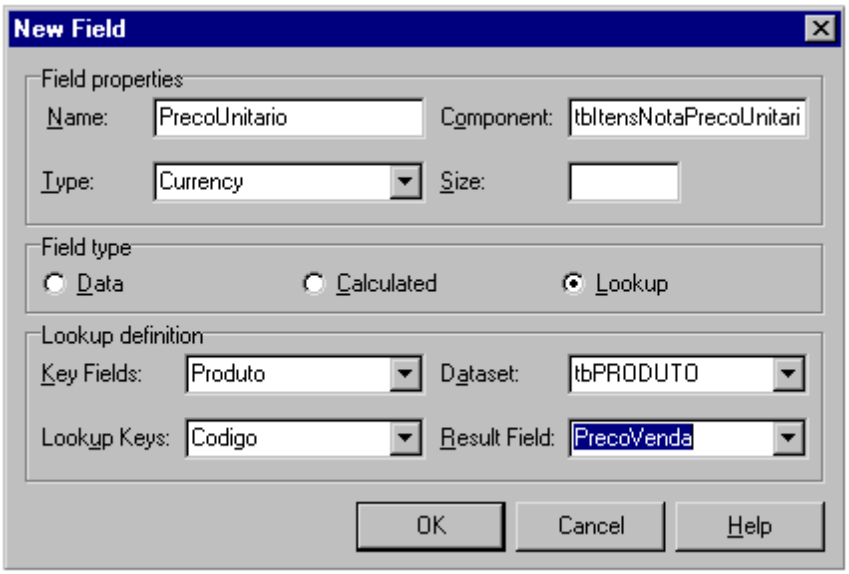
Este tipo de campo "virtual" armazena o valor de um campo que está em outra tabela. A busca desse valor é feita através de pesquisa utilizando a chave primária. Vamos criar um campo chamado **PrecoUnitario** na tabela **tbItemNota** cujo valor que será "trazido" da tabela **tbPRODUTO** através do código do produto.

❑ PASSO 1

Dê um duplo-clique na tabela **tbItemNota**. Aparecerá uma pequena janela contendo os objetos-campo da tabela. Sobre essa janela clique o botão direito do mouse e escolha a opção ***New field...***.

❑ PASSO 2

Na janela que será apresentada, preencha as seguintes informações:



Name	Nome do novo campo "virtual" que será criado na tabela.
Component	Nome do objeto-campo que será criado.
Type	Tipo do novo campo "virtual".
Key Fields	Campo que será utilizado na pesquisa.
Dataset	Tabela onde está o campo desejado.
Lookup key	Chave primária da tabela onde está o campo desejado
Result Field	Nome do campo cujo valor se deseja "trazer".

O atributo **Component** é preenchido automaticamente mas pode ser alterado, se preferir. Este nome é usado para referenciar o objeto-campo nos programas do sistema.

- **PASSO 3**

Clique no botão **OK** e pronto. É criado um novo campo na tabela. Este novo campo não está fisicamente gravado no disco. Ele é mantido automaticamente pelo sistema e seu valor é trazido do campo **PreçoVenda** da tabela de **Produto** (tbProduto).

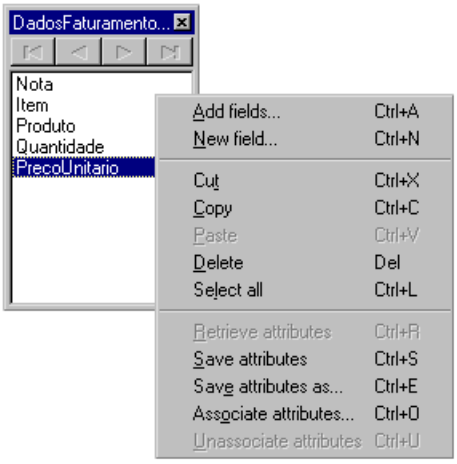
Criando campos calculados

Outro tipo de campo "virtual" é o campo calculado. Um campo calculado armazena o resultado da uma operação realizada com alguns campos da tabela.

Utilizando também a tabela **tbItemNota** como exemplo, poderíamos pensar que nos pouparia muito trabalho se tivéssemos armazenado na própria tabela o resultado da multiplicação do **PrecoUnitario** com a **Quantidade**.

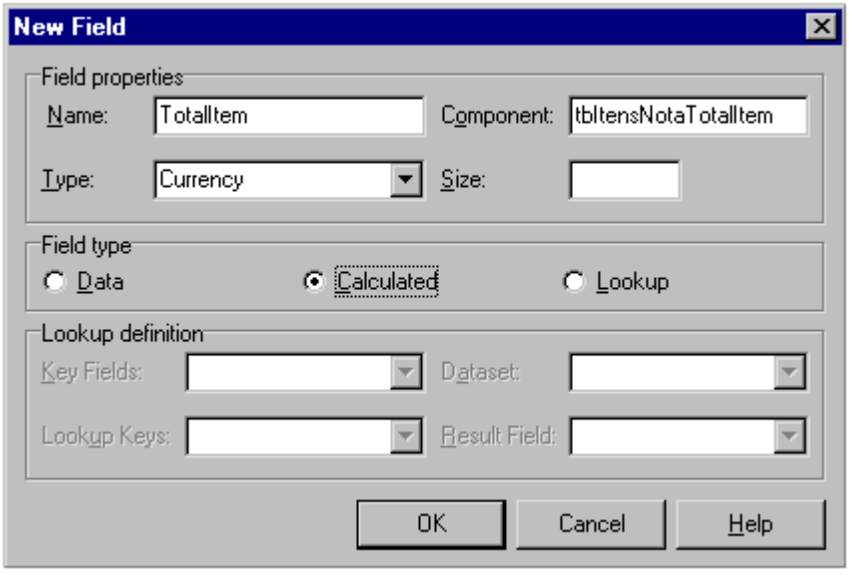
❑ PASSO 1

Dê um duplo-clique na tabela **tbItemNota**. Aparecerá uma pequena janela contendo os objetos-campo da tabela. Sobre essa janela clique o botão direito do mouse e escolha a opção ***New field...***.



❑ PASSO 2

Na janela que será apresentada, preencha as seguintes informações:



Name	Nome do novo campo "virtual" que será criado na tabela.
Component	Nome do objeto-campo que será criado.
Type	Tipo do novo campo "virtual".

- **PASSO 3**

Clique no botão OK e será criado um novo campo na tabela que não está fisicamente gravado no disco. Porém, o valor deste campo não é mantido automaticamente pelo sistema. É necessário definir na tabela **tbItemNota** o evento **OnCalcFields**.

Selecione a tabela **tbItemNota** e na janela **Object Inspector** clique na página **Events**. No evento **OnCalcFields** digite **TotalItem** e pressione **Enter**. Será criado uma **procedure** onde deverá ser digitada apenas uma linha de código.

```
procedure TDadosFaturamento.TotalItem(DataSet: TDataSet);
begin
  tbItensNotaTotalItem.Value    :=    tbItensNotaQuantidade.Value    *
 tbItensNotaPrecoUnitario.Value;
end;
```

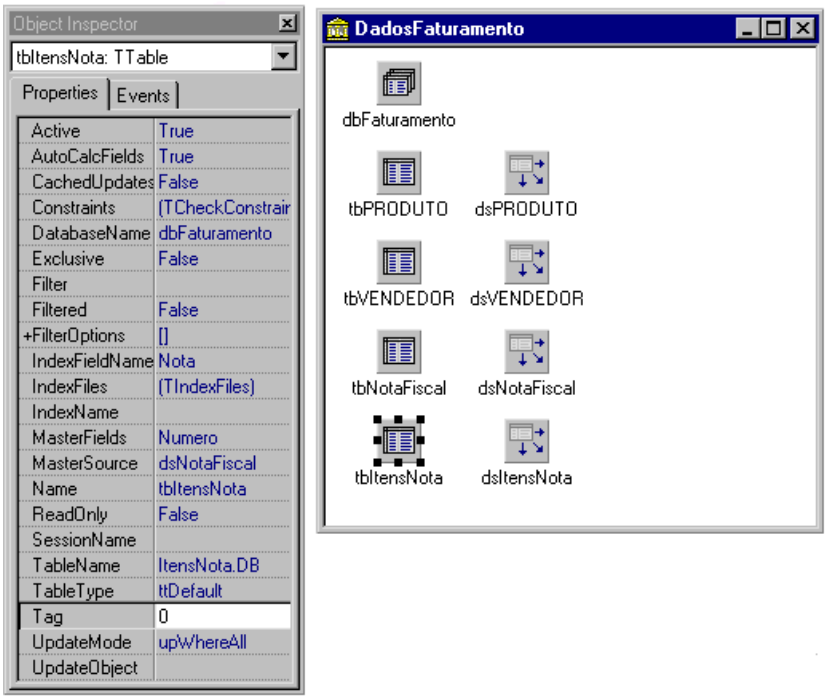
7

Relacionando tabelas

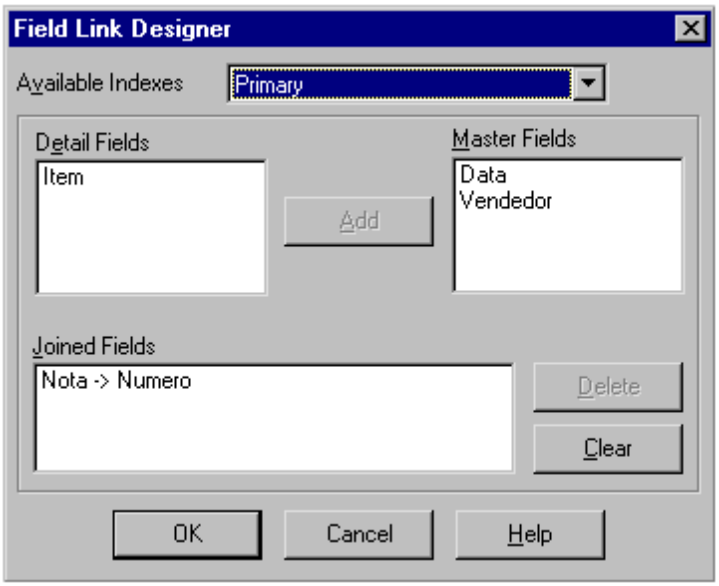
Existe um relacionamento 1 para N entre as tabelas **tbNotaFiscal** e a tabela **tbItemNota**. Uma nota fiscal possui vários itens. Este tipo de relacionamento é chamado pelo Delphi de **Master-Detail**.

No nosso exemplo a tabela *Master* é a tabela **tbNotaFiscal** e a tabela *Detail* é a tabela **tbItemNota**. Os relacionamentos entre as tabelas é feito através de um ou mais campos. No caso, o campo **Nota** da tabela *Detail* (**tbItemNota**) deve se relacionar com o campo **Numero** da tabela *Master* (**tbNotaFiscal**);

Para se definir um relacionamento Master-Detail basta alterar alguns atributos da tabela *Detail*.



Na propriedade **MasterFields** defini-se o campo de relacionamento. Esta definição é feita utilizando-se o **Field Linker Designer**.



8

Criando a janela Principal

O primeiro formulário (janela) que iremos criar é o formulário principal. O formulário principal terá apenas o menu principal de nossa aplicação. Este menu servirá para guiar a implementação de todos os demais formulários do sistema.

❑ PASSO 1

Para criar um novo formulário selecione a opção do menu ***File|New Form***. Será criado um formulário que inicialmente terá o nome Form1.

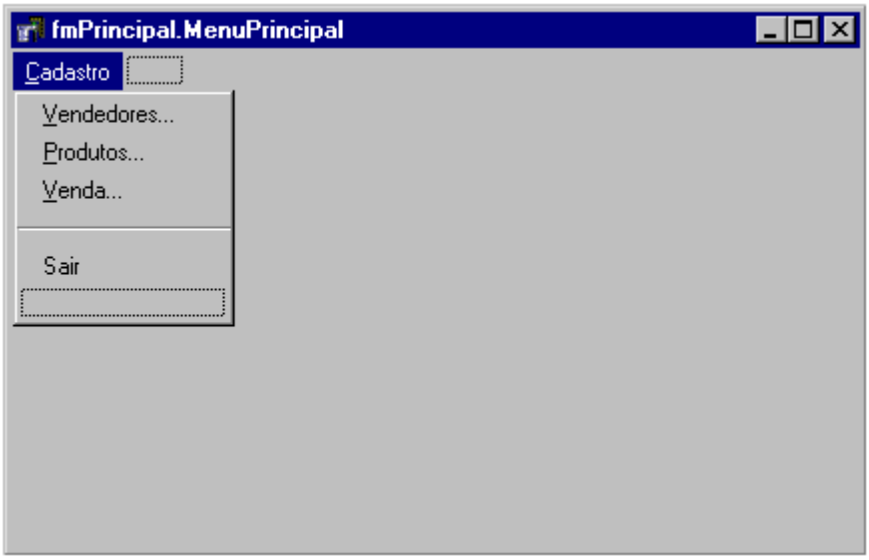
Altere as seguintes propriedades da janela.

Properties	
Caption	SisFat 1.0
Name	FmPrincipal
Position	PoScreenCenter

❑ PASSO 2

Ponha um componente **MainMenu** (página Standard) em qualquer posição e altere a propriedade **Name** para **MenuPrincipal**.

Dê um duplo-clique sobre esse componente e será apresentado um **Editor de Menu**, onde você poderá definir as opções do menu.



Cada item do menu é um objeto, possuindo suas propriedades e eventos. Para cada item do menu, altere as seguintes propriedades:

Caption	Name
&Cadastro	MenuCadastro
&Vendedores	OpcaoVendedores
&Produtos	OpcaoProduto
&Venda	OpcaoVenda
-	N1
Sair	OpcaoSair

❑ PASSO 3

Os eventos deste menu serão definidos assim que as janelas correspondentes forem criadas. Podemos, no entanto, definir o evento para a opção **Sair**. Para isso dê um duplo-clique sobre esta opção. Automaticamente será criada uma Procedure para que você escreva o código (programa) que será executado na seleção desta opção.

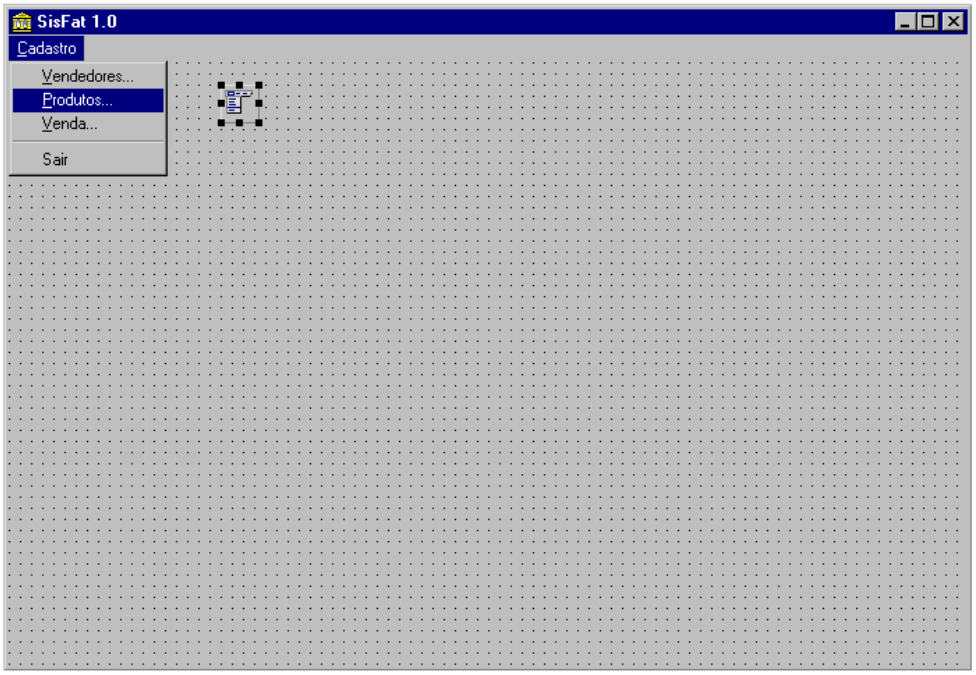
```
procedure TfmPrincipal.OpcaoSairClick(Sender: TObject);
begin
   Close;
end;
```

Digite apenas o comando **Close;** na linha onde o cursor está posicionado.

❑ PASSO 4

Salve o formulário utilizando o comando ***File|Save As***. O nome do arquivo será **Principal**, e será gravado na pasta de programas (**C:\SisFat\Programas**).

O Delphi, na verdade, irá gravar dois arquivo. Um arquivo com extensão **.pas** e um outro arquivo com extensão **.dfm** (**D**elphi **F**or**m**).



9

Criando a janela "Vendedores"

Vamos agora criar o formulário para entrada de dados da tabela Vendedores. Existe uma forma automática de se criar formulários de banco de dados utilizando-se o **Database Form Wizard**. Iremos, porém, criar os formulários de forma manual.

❑ PASSO 1

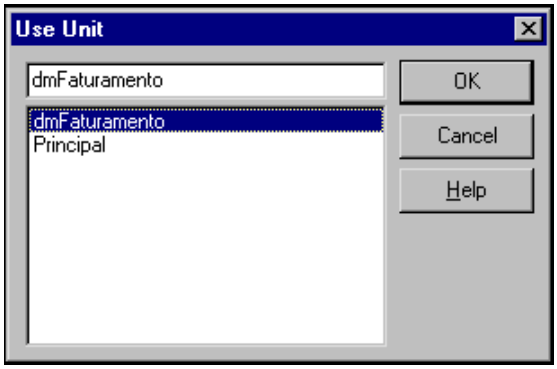
No menu, selecione ***File|New Form***. Será apresentado um formulário em branco. Altera as seguintes propriedades:

Properties	
Caption	Vendedores
BorderStyle	bsSingle
Name	fmVendedor
Position	poScreenCenter

Antes de tudo, salve o formulário (***File|Save As...***) com o nome **Vendedor**.

❑ PASSO 2

Este formulário, assim como todos os formulários que fazem atualização de tabelas, vai referenciar uma ou mais tabelas que estão definidas no **Data Module** da aplicação (**dmFaturamento**). Para que este formulário possa acessar tais tabelas, é necessário declarar este relacionamento. Para isso, selecione a opção ***File|Use Unit*** do menu principal. Será apresentada a seguinte janela:



Selecione **dmFaturamento** e clique OK.

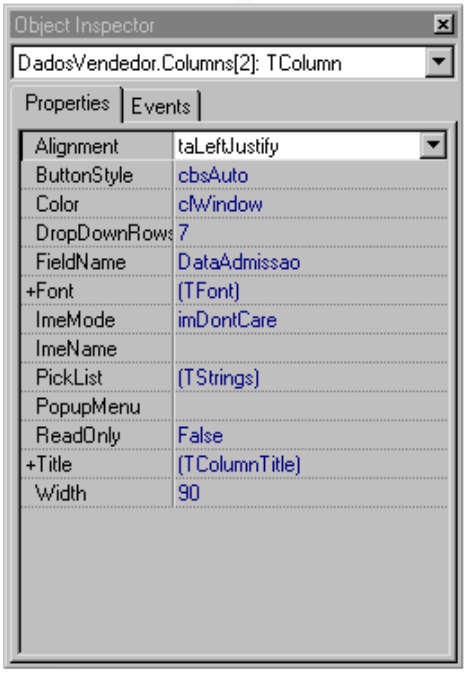
❑ PASSO 3

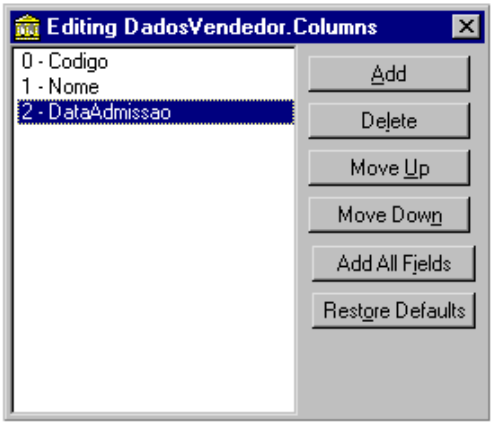
Inclua no formulário um componente **DBGrid** (**Data Controls**).

Altere algumas de suas propriedades:

Properties	
DataSource	DadosFaturamento.dsVENDEDOR
Name	DadosVendedor

Selecionando a propriedade **Columns** (ou dê um duplo-clique no componente DBGrid) é apresentado um "Editor de Colunas". Através dele é possível alterar propriedades e eventos de cada coluna individualmente.



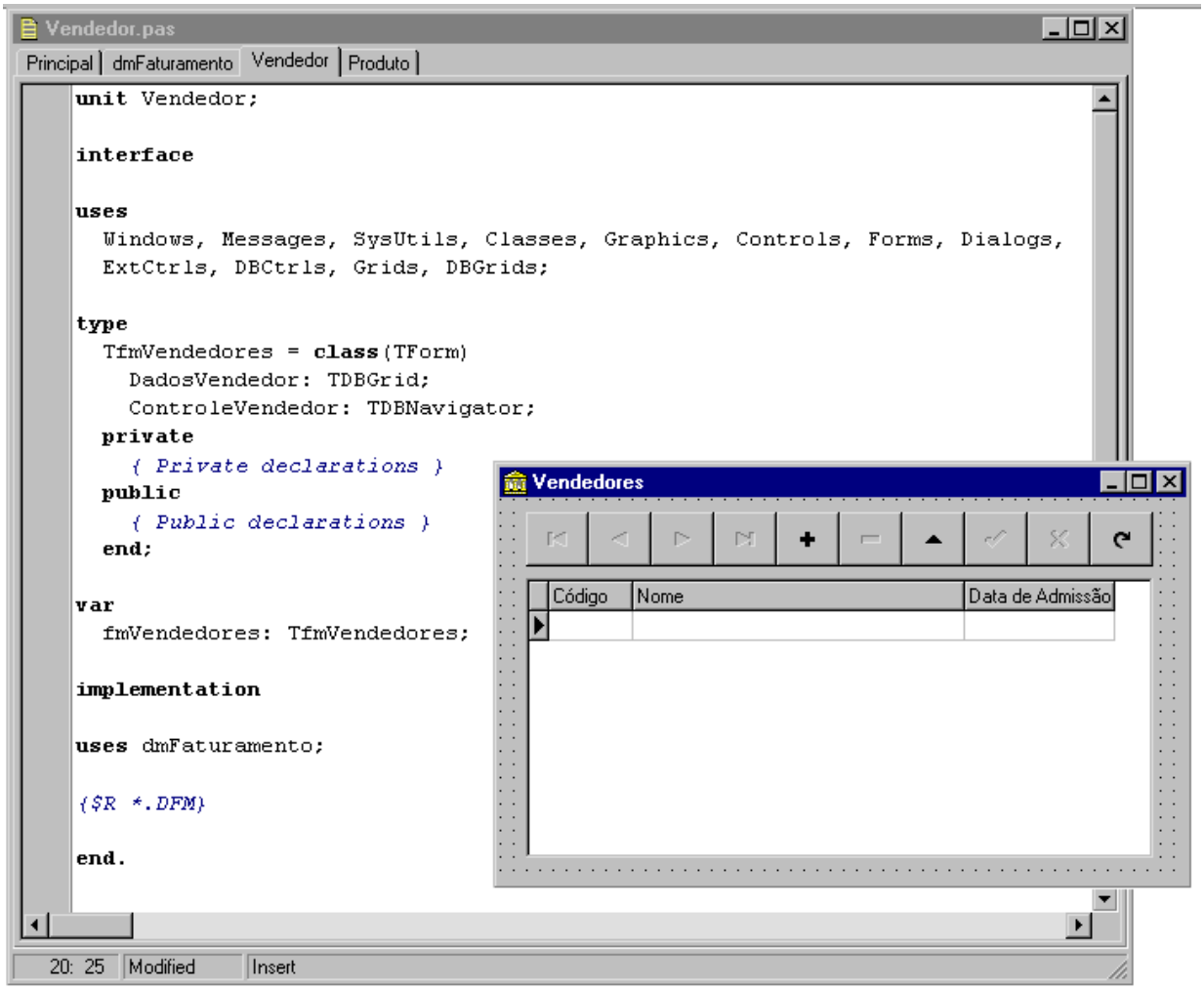


❑ PASSO 4

Inclua no formulário um componente **DBNavigador**.

Altere as seguintes propriedades:

Properties	
DataSource	DadosFaturamento.dsVENDEDOR
Name	ControleVendedor



10

Criando a janela “Produtos”

Seguindo o exemplo da criação da janela “Vendedores”, é criada a janela “Produtos”

- Criar novo formulário (***File|New Form***)com as seguintes propriedades:

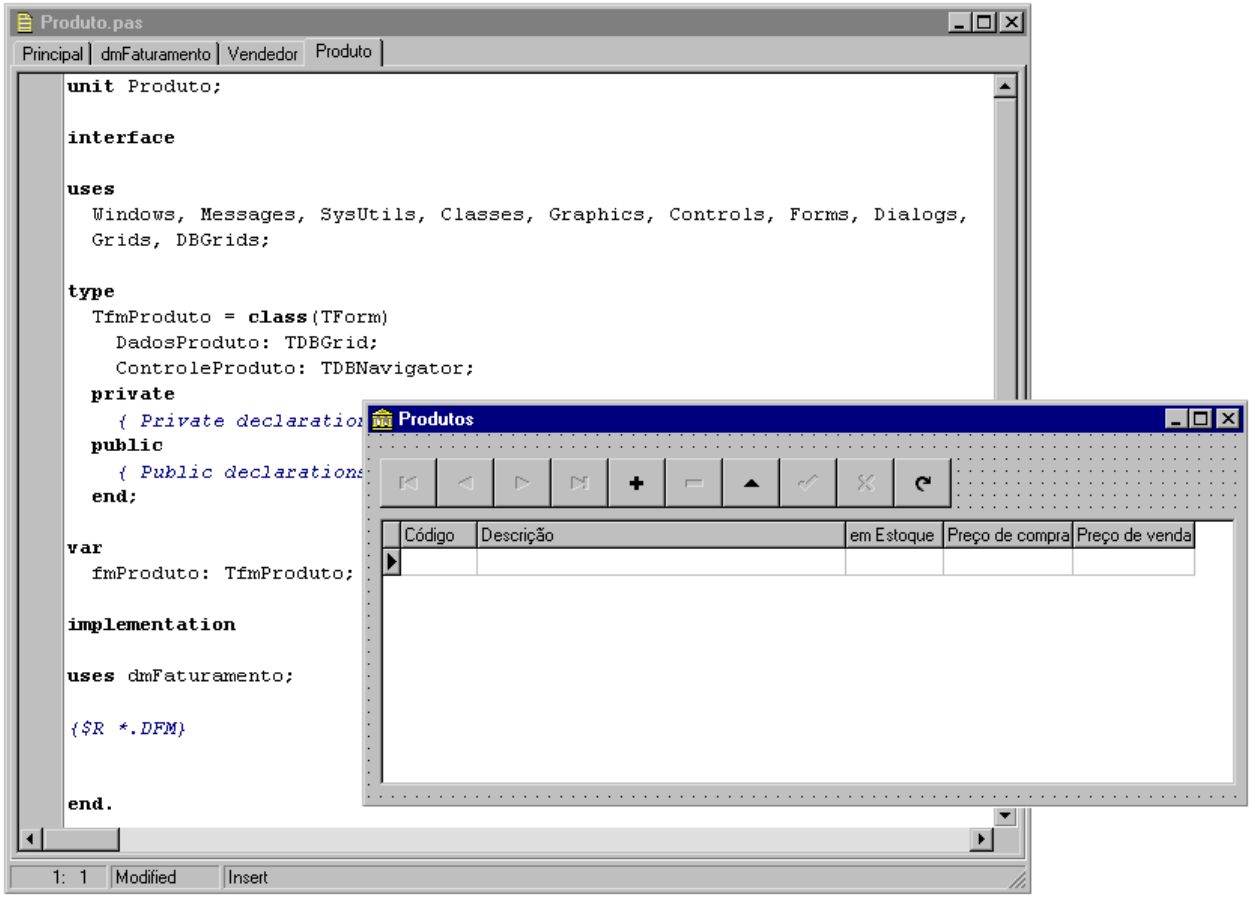
Properties	
Caption	Produtos
BorderStyle	bsSingle
Name	fmProduto
Position	poScreenCenter

- Salvar este formulário com o nome **Produtos**
- ***File|Use Unit*** → **dmFaturamento**
- Incluir no formulário um componente **DBGrid** (**Data Controls**) e alteraras seguintes propriedades:

Properties	
DataSource	DadosFaturamento.dsPRODUTO
Name	DadosProduto

- Configurar cada coluna individualmente;
- Inclua no formulário um componente **DBNavigador** e altere as seguintes propriedades:

Properties	
DataSource	DadosFaturamento.dsPRODUTO
Name	ControleProduto

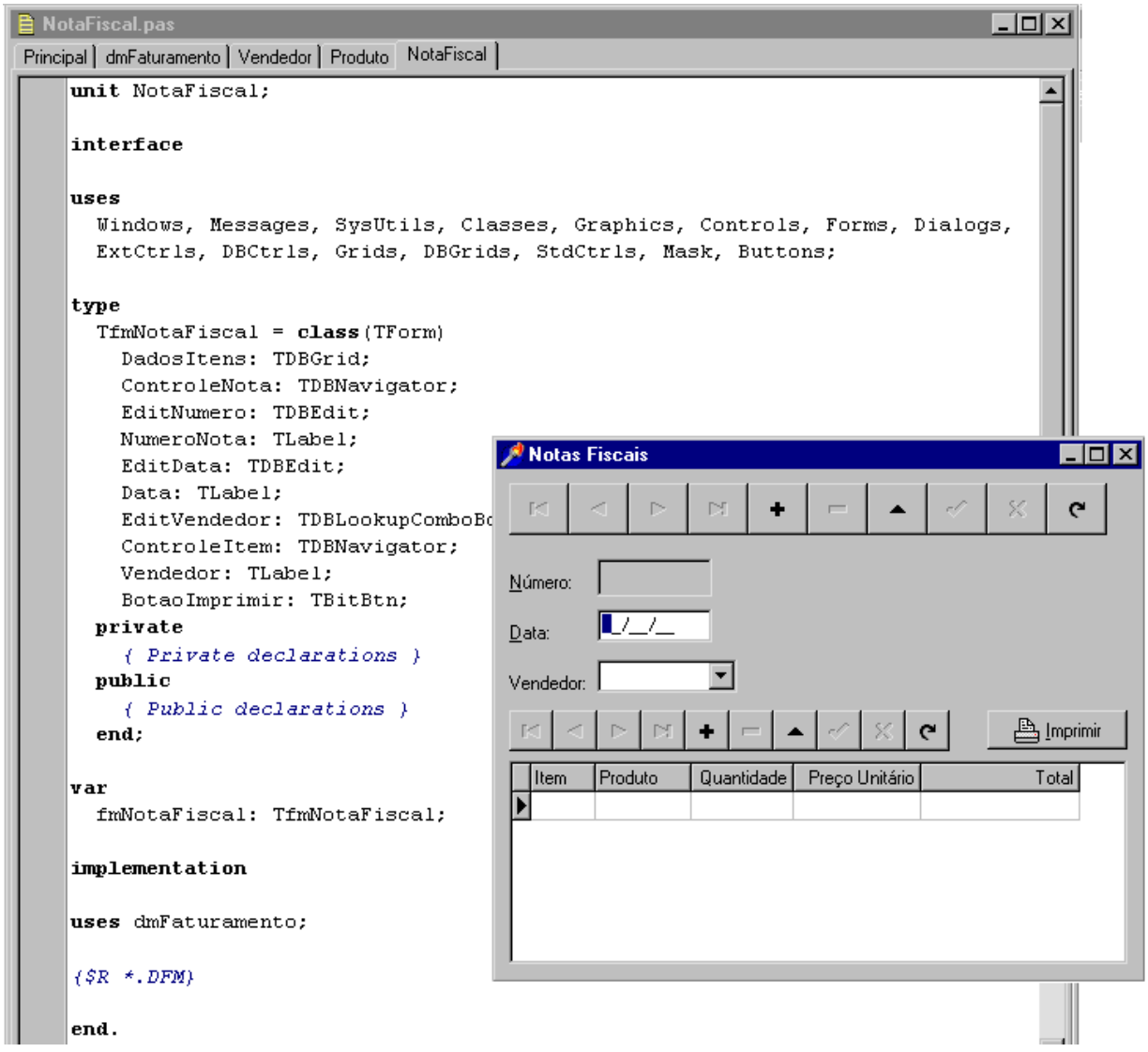


11

Criando a janela “Notas fiscais”

O formulário de entrada de dados das notas fiscais terá as seguintes propriedades:

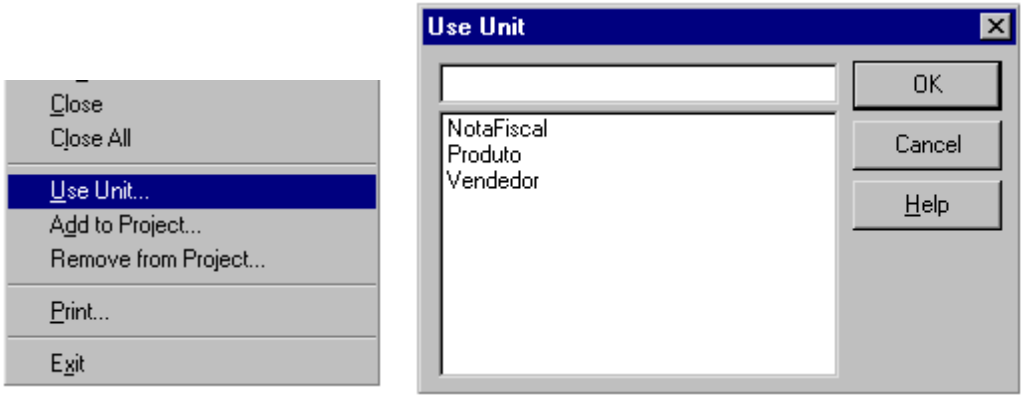
Properties	
Caption	Notas Fiscais
BorderStyle	bsSingle
Name	fmNotaFiscal
Position	poSreenCenter



12

Completando a janela principal

O formulário principal irá acessar os três formulários de entrada de dados que foram criados (**fmVendedor**, **fmProduto**, **fmNotaFiscal**). Para que o formulário principal consiga referenciar cada um deste formulários, é necessário selecioná-los através da opção ***File|Use Unit*** selecione.



Falta agora completar o menu principal definindo-se o evento **OnClick** de cada opção. Para isso, basta clicar em cada opção do menu e definir a linha de código para a chamada da janela correspondente.

Após a definição de todas as procedures, o formulário principal terá o seguinte código:

```
var
  fmPrincipal: TfmPrincipal;

implementation

uses dmFaturamento, Vendedor, Produto, NotaFiscal;

{$R *.DFM}

procedure TfmPrincipal.OpcaoSairClick(Sender: TObject);
begin
   Close;
end;

procedure TfmPrincipal.OpcaoVendedoresClick(Sender: TObject);
begin
   fmVendedor.ShowModal;
end;

procedure TfmPrincipal.OpcaoProdutosClick(Sender: TObject);
begin
  fmProduto.ShowModal;
end;

procedure TfmPrincipal.OpcaoVendaClick(Sender: TObject);
begin
  fmNotaFiscal.ShowModal
end;

end.
```

13

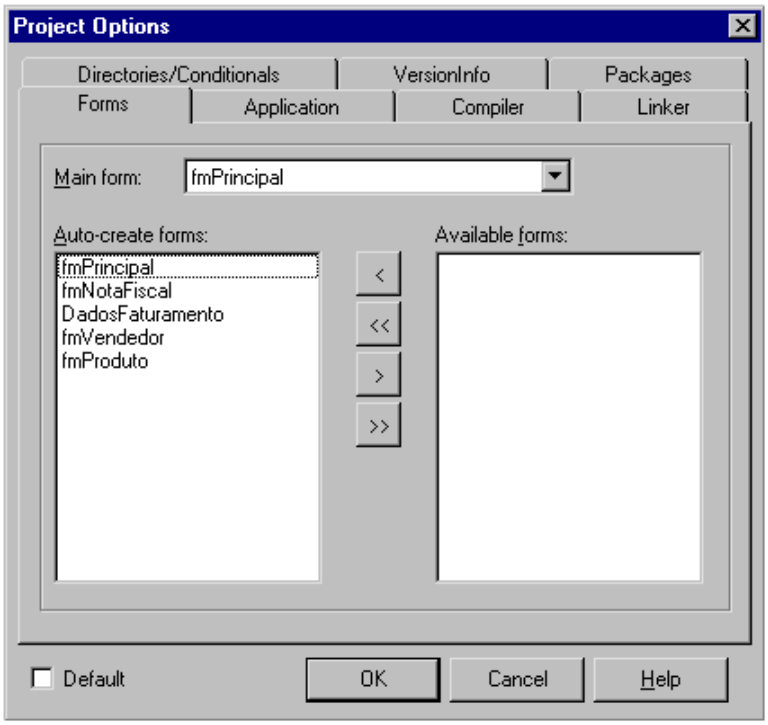
Configurando o Projeto

Para configurar a aplicação utilize a opção do menu do Delphi ***Project|Options...*** Apesar de existir várias páginas, as páginas que iremos ver são as páginas **Forms** e **Application**.

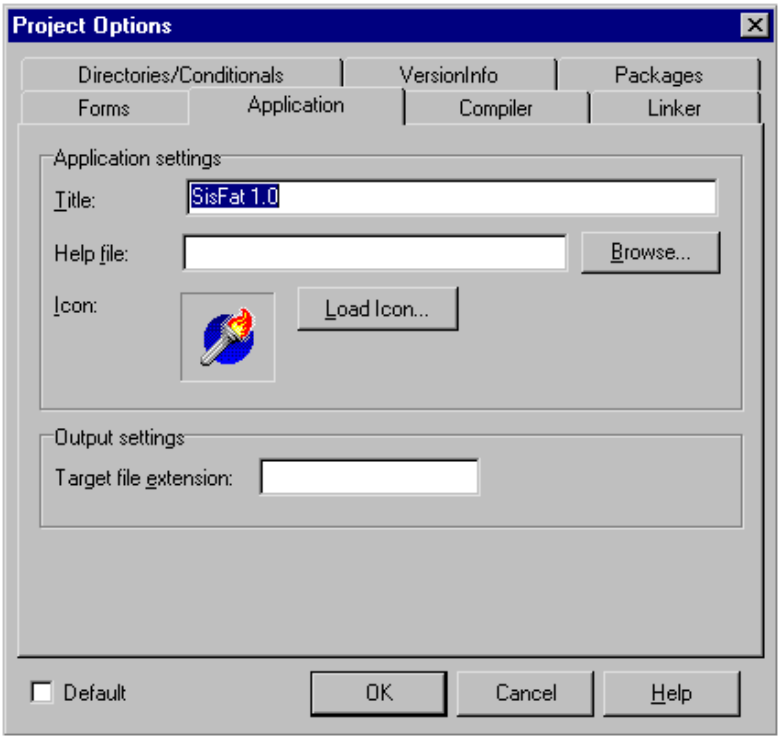
Na página **Forms**, no campo **Main form**, você define o formulário principal do projeto. Isto é, a janela que será apresentada quando você executar a aplicação. No nosso exemplo, o formulário que contém o menu, ou seja **fmPrincipal**.

Na janela **Auto-create forms:** aparecem os formulários que são criados automaticamente, ao ser iniciada a aplicação. Pode-se dizer que estes formulários ocupam memória, mesmo que não estejam visíveis.

Na janela **Available forms:** estão os formulários que serão criados em tempo de execução. Esses formulários não ocupam memória, até que eles sejam apresentados na tela. Porém, o processo de alocação e liberação de memória para esses formulários é de responsabilidade do programador.



Na página **Application** são configurados o título da aplicação (**Title:**), o arquivo de help que será utilizado pela aplicação (**Help file:**) e o ícone para representar a aplicação (**Icon**:)


Project Options
Directories/Conditionals
VersionInfo
Packages
Forms
Application
Compiler
Linker
Application settings
Title:
SisFat 1.0
Help file:
Browse...
Icon:
Load Icon...
Output settings
Target file extension:
Default
OK
Cancel
Help