



Delphi e Object Pascal

Autor:
Maurício Capobianco Lopes



UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO

DISCIPLINA: Linguagem de Programação Visual
PROFESSOR: Maurício Capobianco Lopes

SUMÁRIO

Programação Orientada a Objetos (POO)

- Classes

- Objetos

- Herança

- Encapsulamento

- Polimorfismo

- DELPHI e a POO

 - Qualificadores de Acesso

 - Construtores e Destrutores

 - Implementação de Classes

Biblioteca de Classes

- TApplication

- TScreen

- TPrinter

Criação de Componentes

- O que é um componente

- Componentes: vantagens e desvantagens

- Quando construir um componente

- Criando um Componente

- Instalando o Componente na Biblioteca de Classes

- Criando um ícone personalizado para um componente

- Adicionando Propriedades ao Componente

- Adicionando Eventos ao Componente

PROGRAMAÇÃO ORIENTADA A OBJETOS (POO)

A **Programação Orientada a Objetos** baseia-se na definição de objetos ou classes do mundo real.

Um **Objeto** é uma entidade, ou modelo da vida real. Uma **Classe** é um modelo a partir do qual os objetos são criados.

Para ser definida como orientada a objetos uma linguagem de programação deve suportar, no mínimo, as seguintes **características**:

- **Herança**: uma classe-objeto pode herdar características de outra classe-objeto visando a reutilização de código;
- **Encapsulamento**: uma classe-objeto não pode ver detalhes de implementação de outra classe-objeto.
- **Polimorfismo**: uma classe-objeto deve discernir, dentre métodos homônimos, o que deve ser executado.

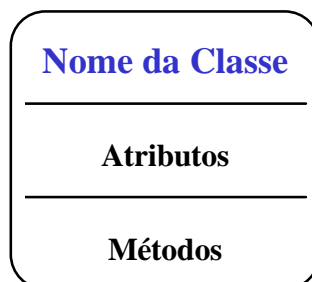
A maneira como o DELPHI permite implementar e manipular estas características pode ser visualizada no item **DELPHI e a POO**.

CLASSES

Classes são “moldes” ou **abstrações** de objetos. Uma classe descreve um conjunto de objetos semelhantes, sendo definida por seus **Atributos** e **Métodos**:

- **Atributos**: são os dados ou informações do estado de um objeto;
- **Métodos**: são as operações efetuadas pelos objetos.

REPRESENTAÇÃO DE UMA CLASSE



OBJETOS

Objetos são modelos que representam entidades do mundo real. Objetos são instâncias de uma classe, ou seja, quando se define uma classe preenchendo-se o seu estado (Atributos) e implementando seu comportamento (Métodos).

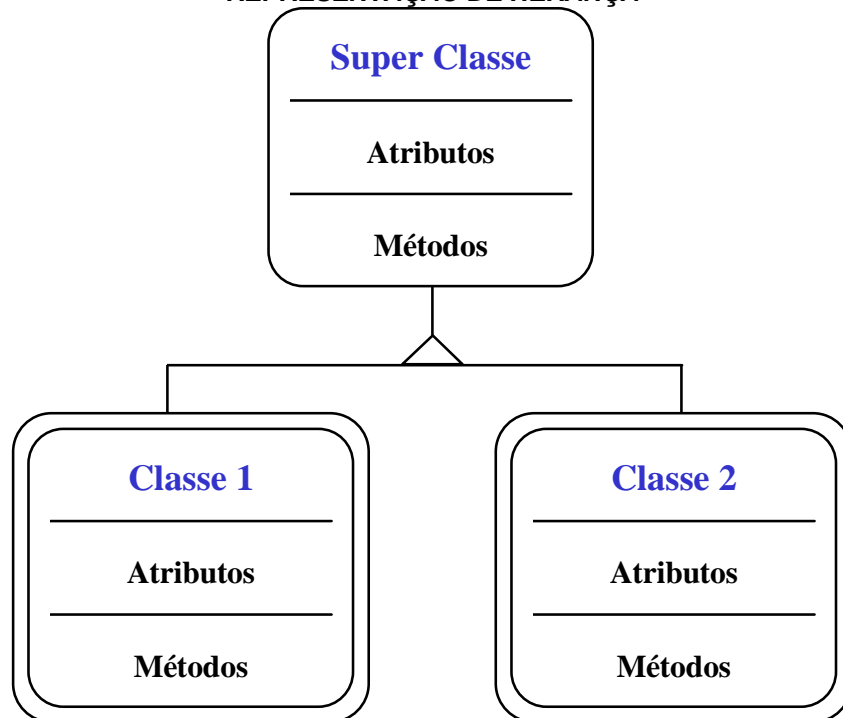
REPRESENTAÇÃO DE UM OBJETO



HERANÇA

Herança é a capacidade das classes-objetos de compartilhar atributos e métodos. Na herança a classe filha herda as características da classe pai (superclasse).

REPRESENTAÇÃO DE HERANÇA

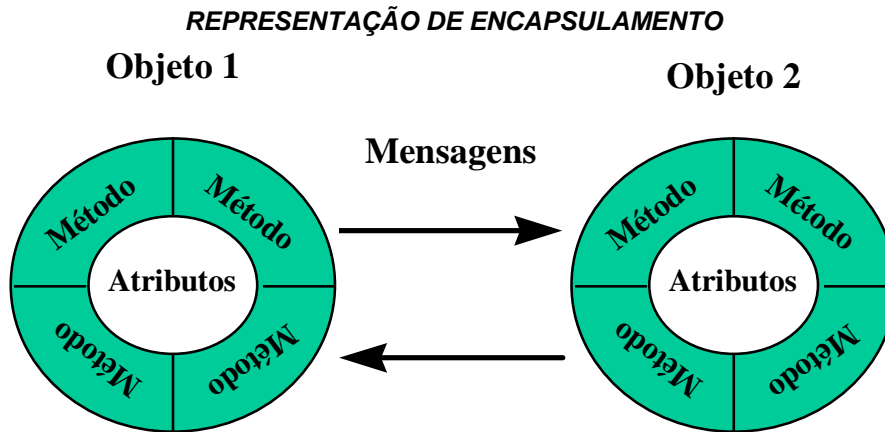


ENCAPSULAMENTO

Encapsulamento é a capacidade das classes-objetos de se comunicarem com outras classes-objetos através de sua interface, independente de sua implementação interna.

Interface é a forma como a classe-objeto se relaciona com as demais classes-objetos do sistema.

Na programação orientada a objetos é desejável que a comunicação (**mensagens**) entre os objetos seja feita apenas através dos métodos, ou seja, que outros objetos não possam acessar diretamente os atributos do objeto.



POLIMORFISMO

Polimorfismo é a capacidade de uma classe-objeto responder da maneira mais apropriada para sua classe.

Com o polimorfismo não é necessário saber exatamente qual o tipo do objeto para o qual se está enviando uma mensagem, bastando saber apenas que a mensagem faz parte do protocolo de sua classe.

O principal **objetivo** do polimorfismo é a construção de aplicações genéricas.

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:

1. A classe mãe é denominada **TObject**. Esta classe deve ser utilizada quando a classe não tiver bem definida sua classe mãe.
2. A definição de uma classe deve ser feita na área de **interface** de uma UNIT.

3. 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. Sendo assim, os 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:

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

* Esta definição não está completa pois faltam os qualificadores *published* e *automated* que serão estudados na criação de componentes.

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 convencionou-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-objeto.

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 : string; Endereco : string);
```

end;

implementation

constructor TPessoa.Create;

begin

inherited Create;

 Nome := '';

 Endereco := '';

end;

destructor TPessoa.Destroy;

begin

inherited Destroy;

end;

procedure TPessoa.AtualizarDados (Nome : string; 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.

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** (no menu principal acesse *View - Browser*).

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

CLASSES VISÍVEIS

As classes visíveis são aquelas que podem ser manipuladas através da **paleta de componentes**.

CLASSES NÃO-VISÍVEIS

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

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 *form* principal da aplicação;
- **Title^{*}**: contém o título da aplicação quando a mesma está minimizada;
- **HelpFile^{*}**: armazena o nome do arquivo de *Help* da aplicação;
- **Icon^{*}**: contém o ícone que representa a aplicação.

TSCREEN

^{*} *Runtime only*: só pode ser usada no código do programa, ou na opção do menu *Project - Options*.

[#] *Read-only*: propriedade somente de leitura. Não pode ser alterada no código do programa.

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 *forms* 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**.

^{*} *Runtime only*: só pode ser usada no código do programa, ou na opção do menu *Project - Options*.

[#] *Read-only*: propriedade somente de leitura. Não pode ser alterada no código do programa.

CRIAÇÃO DE COMPONENTES

O QUE É UM COMPONENTE

Em Delphi, um componente é uma classe de uma aplicação que pode ser manipulada visualmente pelo programador. Os componentes são classes descendentes da classe **TComponent**.

Os componentes podem ser especializações de outros já existentes ou possuir funcionalidade e estrutura totalmente novas.

COMPONENTES: VANTAGENS E DESVANTAGENS

As principais **vantagens** da criação de componentes são:

- funcionam como bibliotecas de programas;
- ficam disponíveis na paleta de componentes;
- ficam incorporados ao código fonte da aplicação.

As principais **desvantagens** são:

- o programador deve conhecer profundamente a **VCL** (*Visual Component Library*);
- devem pressupor a inexistência de erros no código fonte;
- requerem um grande esforço de programação.

QUANDO CONSTRUIR UM COMPONENTE

Um componente deve ser construído quando:

- existe tempo hábil para fazê-lo;
- não existe outro componente na **VCL** capaz de realizar as funções desejadas;
- sua utilização será feita de forma habitual.

Para construir um componente que seja a especialização de um outro, deve-se ter conhecimento de todas as classes ancestrais àquele componente. Os componentes totalmente novos devem ser descendentes da classes **TComponent** que tem apenas as propriedades *Name* e *Tag*.

CRIANDO UM COMPONENTE

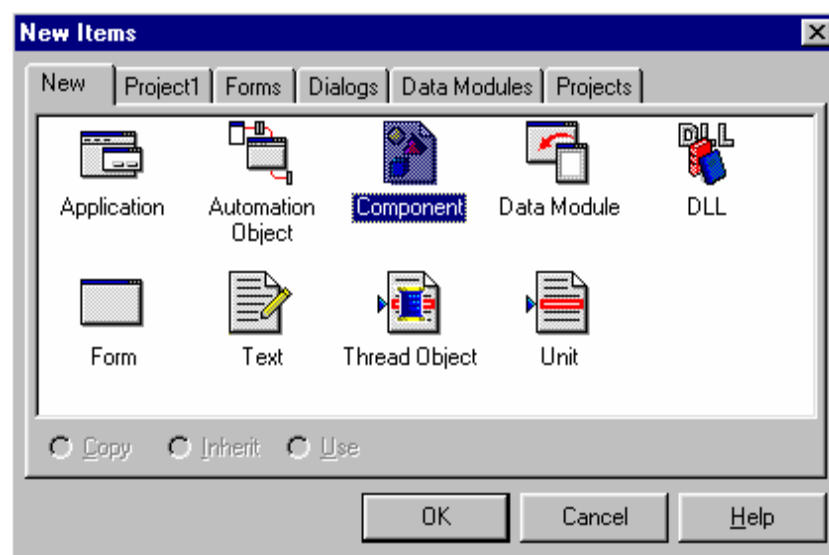
Neste item serão apresentados os principais aspectos da criação de um componente, através de uma sequência de passos que permitirá que se tenha um componente de edição onde possa ser feita apenas a edição de valores numéricos.

O primeiro passo para a criação de um componente especializado é definir sua **classe-mãe**. Neste caso, deseja-se um componente de edição, então de forma natural esperaria-se que sua classe-mãe fosse a classe **TEdit**. No entanto a classe **TEdit** possui em sua estrutura uma propriedade denominada **Text**, que não é desejada para o novo componente. Neste caso, deve-se verificar na **VCL**, qual seria a classe-mãe mais adequada.

Definiremos como **classe-mãe** do novo componente a classe **TCustomEdit**, que é justamente a classe ancestral de **TEdit**.

A partir daqui será iniciada a construção do novo componente.

1. Clique na opção **File**, no meu principal e em seguida em **New**.
2. Ao abrir a janela **New Items**, selecione a opção **Component** e clique em **OK**

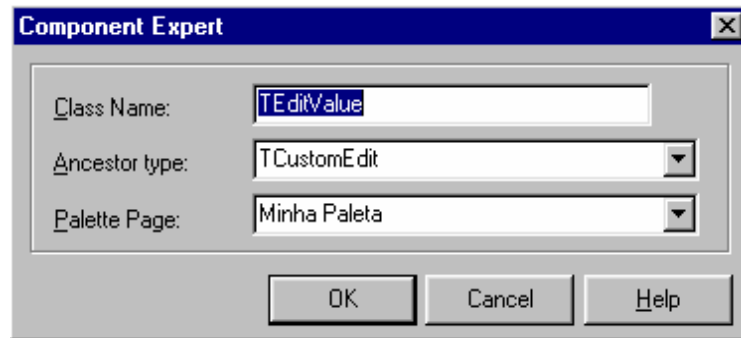


Criando um novo componente

A seguir aparecerá a janela **Component Expert**, onde deve ser definido o posicionamento do componente dentro da **VCL**:

1. Na opção **Class Name** deve ser digitado o nome do novo componente. Para este exemplo nomeie o novo componente com **TEditValue**.

2. Na opção **Ancestor Type** deve ser definida a classe ancestral do componente, Neste caso selecione **TCustomEdit**.
3. Na opção **Palette Page** deve ser colocado o nome da paleta onde aparecerá o novo componente. Neste caso digite "**Minha Paleta**".
4. Clique em **OK**.



Component Expert: definindo uma nova classe

Neste momento deve aparecer na janela **Code Editor** o código fonte do novo componente, que é como qualquer *unit* do DELPHI.

```
unit Unit2;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs,
  StdCtrls;

type
  TEditValue = class(TCustomEdit)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
  published
    { Published declarations }
  end;
procedure Register;

implementation
procedure Register;
begin
  RegisterComponents('Minha Paleta', [TEditValue]);
end;
```

end.

Observe que:

- a) a classe **TEditValue** é classe filha de **TCustomEdit**;
- b) foi incluído, na definição da classe, o qualificador de acesso **published**;
- c) foi incluída, na *implementation*, a **procedure Register**.

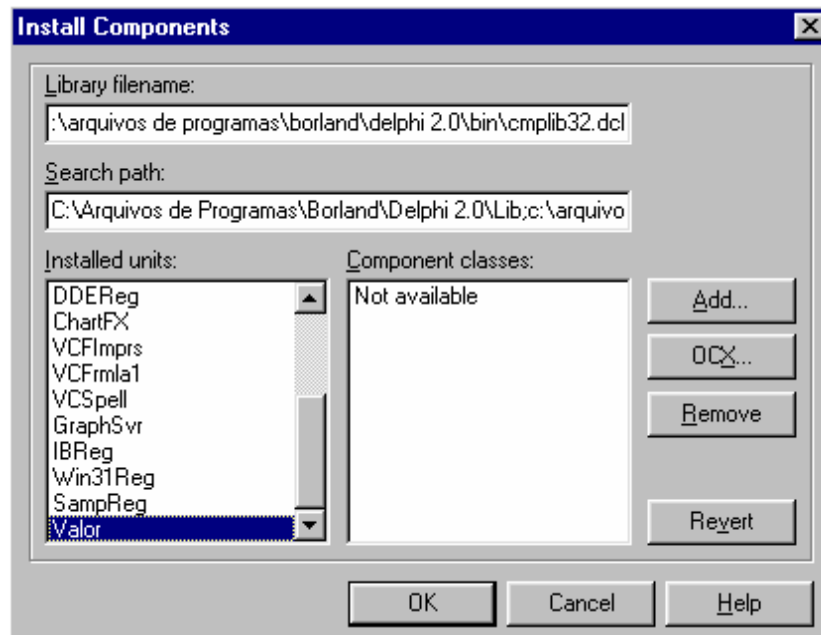
O item b) será avaliado em adicionando propriedades ao componente. Em relação ao item c) note que na **procedure Register** aparecem dois parâmetros: o nome digitado para a paleta e o nome da classe criada. Esta *procedure* justamente relaciona a paleta e os componentes a serem inseridos na mesma.

Grave a *unit* do componente com o nome "**Valor.PAS**". A *unit* pode ser gravada em qualquer diretório.

INSTALANDO O COMPONENTE NA BIBLIOTECA DE CLASSES

Para serem utilizados, os componentes devem estar instalados na Biblioteca de Classes (**VCL**). No caso do DELPHI 2.0 a biblioteca onde estão colocados os componentes é denominada de **CMPLIB32.DCL**. É nesta biblioteca que deve ser instalado o novo componente. Para isto proceda como segue:

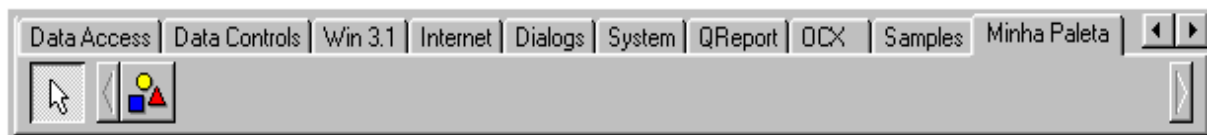
1. Clique no menu **Component** e, em seguida, em **Install**.
2. Na janela *Install Components* clique no botão **ADD**.
3. Selecione o arquivo "**Valor.PAS**", gravado anteriormente, através do botão **Browse** e clique em **OK**. Note que o nome do arquivo aparece no final da coluna *Installed units*.
4. Clique em **OK**.



Instalando um componente na biblioteca

Neste momento a biblioteca de componentes do DELPHI é recompilada. Para isto ele automaticamente fecha todos os arquivos abertos, compila a biblioteca, e reabre os arquivos ao final do processo. Os erros de compilação que porventura ocorrerem são indicados na janela **Messages**.

A partir deste momento deverá aparecer, na paleta de componentes, a “**Minha Paleta**”, contendo o novo componente criado.



Meu primeiro componente

Clique sobre o componente e o coloque em um *Form*. Este componente é um componente de edição, no entanto ele não tem todas as propriedades e os eventos do **TEdit**. Observe que no *Object Inspector* aparecem poucas propriedades e nenhum evento para o componente. A inclusão de propriedades e eventos será discutida no item [adicionando propriedades ao componente](#) e [adicionando eventos ao componente](#).

O ícone que representa o componente é um ícone padrão do DELPHI. No entanto ele pode ser [personalizado](#).

CRIANDO UM ÍCONE PERSONALIZADO PARA UM COMPONENTE

A criação de um ícone personalizado deve ser feita através do programa **Image Editor** que acompanha o DELPHI. Sendo assim, siga os passos abaixo:

1. A partir da pasta **Borland Delphi 2.0** abra o arquivo **Image Editor**.
2. No menu principal clique em **File** e posicione o mouse sobre a opção **New**. O arquivo que representa um componente é denominado de **Delphi Component Resource File** e tem extensão **DCR**.
3. Clique portanto sobre a opção **Delphi Component Resource File**.
4. Na janela *Untitled.dcr* clique, com o botão direito do mouse, sobre a palavra **Contents**.
5. Posicione o mouse sobre a opção **New** e clique em **Bitmap**. O *bitmap* que representa o ícone do componente tem que ter um tamanho de **24 x 24**.
6. Na janela *Bitmap Properties* digite 24 na opção **Width** e 24 na Opção **Height**. Clique em **OK**. Neste momento deverá aparecer na tela um árvore contendo os nomes **Bitmap** e **Bitmap1**.
7. Dê um duplo clique sobre a opção **Bitmap1** e desenhe uma representação para o componente.
8. Após desenhar o *bitmap*, minimize ou feche a janela de desenho e com o botão direito do mouse clique sobre **Bitmap1**.
9. Selecione a opção **Rename**. Mude o nome do *bitmap* para o mesmo nome que foi definido para a classe (no exemplo **TEditValue**).
10. Salve o arquivo com o mesmo nome da *unit* do componente.

Para que o ícone apareça na paleta de componentes é necessário recompilar a biblioteca de classes.

ADICIONANDO PROPRIEDADES AO COMPONENTE

Uma propriedade em um componente deve ser precedida pela palavra **property**. A propriedade pode ter duas características:

- a) **ser pública (public)**: uma propriedade pública só pode ser alterada em tempo de execução, não sendo acessível através do *Object Inspector*.
- b) **ser publicada (published)**: uma propriedade publicada pode ser alterada em tempo de projeto e execução, sendo acessível através do *Object Inspector*.

Uma nova propriedade deve estar associada a um atributo ou método. Para isto deve ser definido o seu método de acesso:

- **read**: o método de acesso read permite verificar o conteúdo de uma propriedade.

- **write**: o método de acesso write permite definir o conteúdo de uma propriedade.

Sendo assim, para definir a propriedade **Valor** no componente **TEditValue** deveria se seguir a definição abaixo:

```
type
  TEditValue = class(TCustomEdit)
  private
    { Private declarations }
    FValor : Extended;
  protected
    { Protected declarations }
  public
    { Public declarations }
  published
    { Published declarations }
    property Valor : Extended read FValor write FValor;
  end;
```

Toda propriedade deve ter um **atributo** ou **método** associado para poder ser encapsulado na classe. Por convenção, no DELPHI o atributo que representa uma propriedade é iniciado pela letra F. O atributo deve ter o mesmo tipo da propriedade.

O método **read** deve ser representado por uma **function** que retorna como resultado um valor que é do mesmo tipo da propriedade.

O método **write** deve ser representado por uma **procedure** com um único parâmetro que deve ter o mesmo tipo da propriedade. Este parâmetro deve ser atribuído à propriedade.

Adicione as seguintes propriedades ao componente:

- **Decimais**: usada para definir a configuração das casas decimais;
- **Máximo**: usada para definir o valor numérico mais alto suportado pelo componente;
- **Mínimo**: usada para definir o valor numérico mais baixo suportado pelo componente;
- **Formato**: usado para definir o formato de apresentação do número.

ADICIONANDO EVENTOS AO COMPONENTE

Um evento deve ser associado a um componente do mesmo modo que uma propriedade. No entanto deve-se definir quando ele será disparado. O DELPHI tem tipos de eventos pré-definidos, tais como, TNotifyEvent,

TMouseEvent, TKeyPressEvent. O programador pode definir seus próprios eventos de acordo com sua necessidade e com os parâmetros que devem ser passados para o mesmo.

Para se definir um novo evento deve-se criar um tipo **procedure (parâmetros) of object**, como definido no exemplo abaixo:

```
type
  TExemploEvent = procedure (Sender : TObject; Valor : real) of object;
```

Para que possam ser executados os eventos devem ser ativados. através de sua chamada, como se fosse uma procedure comum. Antes, no entanto, deve-se verificar se existem comandos associados a ele através do método *Assigned*.

Sendo assim a implementação de um evento poderia ficar como no exemplo abaixo:

```
type
  TEditValue = class(TCustomEdit)
  private
    .
    EMaximo : TNotifyEvent;
    .
    procedure TEditValue.SetValor (Valor : Extended);
  published
    property OnMaximo : TNotifyEvent read EMaximo write EMaximo;
    { Published declarations }
  end;

implementation

procedure TEditValue.SetValor;
begin
  .
  if Assigned (EMaximo) then
    EMaximo (Self);
  .end;
end.
```

Adicione os seguintes eventos ao componente:

- **OnMáximo**: ativado quando o componente atinge seu valor máximo;
- **OnMínimo**: ativado quando o componente atinge seu valor mínimo.