## Orientado a objetos ou Baseado em objetos?!

**Mário Leite**

Alguns autores fazem uma distinção muito rígida entre L*inguagens Orientadas a Objetos* e *Linguagens Baseadas em Objetos***.** Pessoalmente não vejo muita importância nessa discussão; mas de qualquer forma podemos analisar esta questão**.** O argumento utilizado pelos promotores desta polêmica é alicerçado numa convenção (talvez *deles* mesmo), que diz o seguinte: "*Uma linguagem é orientada a objetos se ela suporta perfeitamente quatro conceitos básicos:* ***abstração***, ***herança***, ***polimorfismo*** *e* ***encapsulamento***"**.** Desse modo, ainda do ponto de vista *deles*, se determinada linguagem suporte apenas três desses conceitos ou menos, ela seria considerada *baseada em objetos* e não orientada a objetos**.** Sendo assim, o **Microsoft Visual Basic** (até a versão 6.0) estaria nesta situação, pois esta ferramenta não suporta, integralmente, o mecanismo da *herança***.** O mesmo acontece com o **CA-Clipper 5.xx** (hoje o **Harbour**), apaixonadamente defendido pelos autores Marcelo Ferreira e Flavio J. Jarabeck [*Programação Orientada ao Objeto em Clipper 5.0 – Ed. Makron Books, São Paulo, 1991*]. O VB (nas versões *desktop*) ainda que possua a capacidade de criar classes, objetos, encapsular rotinas e prover mecanismos de abstração, não consegue fazer com que uma classe herde propriedades e métodos de uma outra**.** Já o Clipper (a partir da versão 5.0) possui quatro classes de objetos já pré-definidas na linguagem (com propriedades -variáveis exportadas- e métodos), porém, não permitindo que o programador crie outras classes, além de não possibilitar uma abstração mais visível em nível de desenvolvimento**.** Isto torna a linguagem mais susceptível a críticas quanto à sua capacidade de criar programas baseados em objetos**.** Atualmente, a linguagem mais badalada é o **Python**, considerada "Orientada a Objetos"; mas que, na maioria das vezes, os programadores usam menos de 10% deste paradigma em programas implementados nesta linguagem. No máximo, aproveitam o *intelliSense* da ferramenta, digitando um ponto após uma palavra-chave esperando que apareça algum membro da classe para agilizar a criação do código; mas, utilizar TODO o potencial da OO desta linguagem, são poucos...

Conforme foi frisado, esta é apenas uma convenção muito empregada pelos analistas mais radicais da OOP (**O**bject-**O**riented **P**rogramming - Programação Orientada a Objetos) para argumentar de modo mais favorável à sua linguagem preferida**.** Por outro lado, outros autores (também radicais do outro extremo) defendem a tese de que para ser considerada Orientada a Objetos uma linguagem precisa apenas ***encapsular***; isto, tanto o VB quanto até o velho Clipper pode fazer, vai depender do programador. No caso do Delphi não precisamos nos preocupar com toda essa polêmica "barata", pois ele suporta esses quatro conceitos de maneira bem real para o desenvolvedor**.** O Delphi é uma extensão do Object Pascal no ambiente Windows; por isto, quando falamos em objetos, estamos falando (em tese) de um *record* dentro do qual podemos definir procedures e funções, bem como campos**.** Deste modo, pode-se imaginar um objeto como sendo um *container* que guarda não somente dados (campos) mas também ações (rotinas)**.**

Em termos de programação, quando definimos uma classe, estamos na verdade definindo um tipo de dado sobre o quais serão construídas as futuras variáveis**.** A definição de uma classe é sempre a definição de um tipo, e a definição de uma variável é a definição de um objeto**.** Do ponto de vista de codificação, podemos afirmar que um *objeto* é uma *variável* declarada na cláusula **Var** de uma seção da **Unit** do Delphi. A **figura 1** mostra um esquema genérico em **Delphi** para a criação de uma nova classe e a definição de um objeto (*instância*) dessa classe**.** Na **figura 2** observamos uma situação corriqueira das cláusulas **Type** e **Var** de uma *unit* ligada à um formulário; primeiramente é criada a classe **TForm1** a partir da classe **TForm** e em seguida os membros (instâncias de objetos) colocados na interface**.** Podem ser notadas as semelhanças entre as duas figuras; apenas o *Constructor* e o *Destructor* não estão presentes na criação de um formulário, pois ele é criado (quando o programa é executado) e destruído automaticamente pelo Delphi quando for fechado**.** No caso da criação de classes no VB, podemos observar a **figura 3**; foi declarada uma variável **oBj** como sendo de um novo tipo**: NovaClasse**. Essa classe é criada interativamente através do IDE do VB acionando o *menu* **Project/Add Class Module**: assim como seus membros (*propriedades* e *métodos*)**.** Membro1, Membro2 e Membro3 são as propriedades dessa classe (por exemplo: Código, Nome, Telefone do cliente)**.** Então, a instrução **oBj.Membro3 = valor3** poderia atribuir ao campo "Telefone" um novo valor (numa situação de alteração e/ou inclusão de um novo registro na tabela); e uma outra instrução do tipo **oBj.Salvar** poderia ser uma ordem para que a instância **oBj** (da classe **NovaClasse**) executasse o método **Salvar** (definido previamente por ocasião da criação dessa classe)**.**

Portanto, a menos do mecanismo de herança, que apesar de ser importante, não é fundamental, a questão Orienta a Objetos *versus* Baseada em Objetos não deve ser o argumento principal para a escolha da linguagem de implementação no desenvolvimento de um sistema.

---

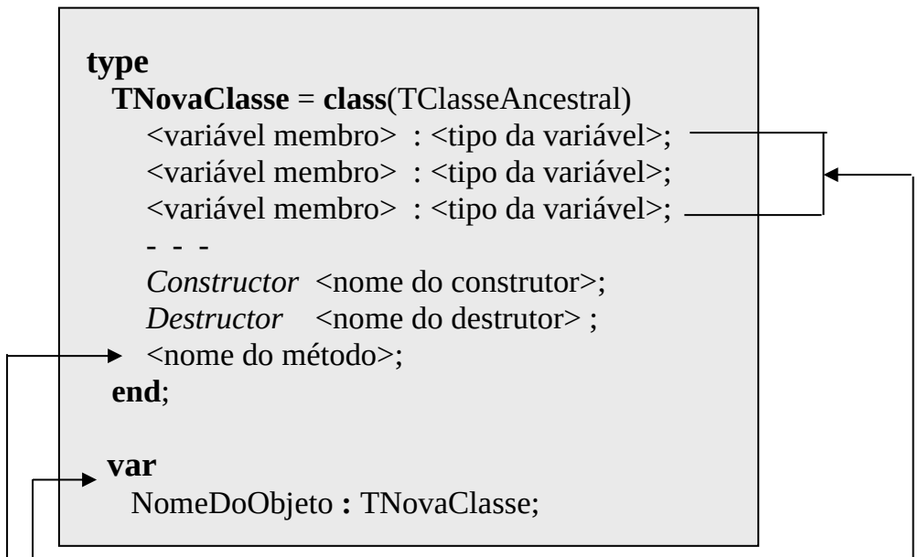


**Figura 1 - Criando uma nova Classe e uma instância dela**

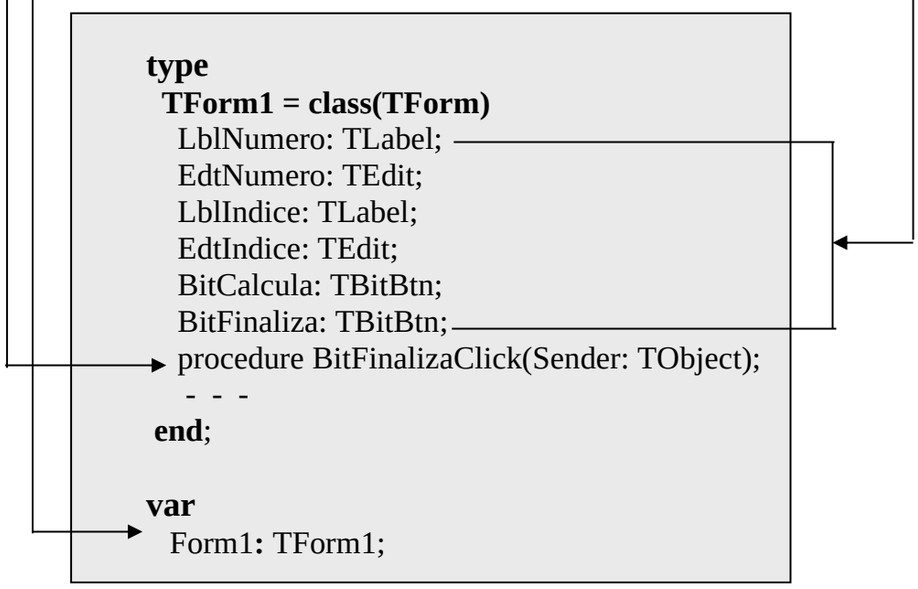


**Figura 2 - Criação do Formulário e seus "membros" agregados**

```
Dim oBj As New NovaClasse

Private Sub Form_Load()
  oBj.Membro1 = valor1
  oBj.Membro2 = valor2
  oBj.Membro3 = valor3
End Sub
```

**Figura 3 - Criação da instância oBj no VB**