OOP com Visualg

*Mário Leite*

...

A Programação Orientada a objetos (**OOP**: **O**bject-**O**riented **P**rogramming em Inglês) remete à uma tecnologia que, embora não seja tão nova assim (criada por Ole-Johan Dahl e Kristen Nygaard na Noruega, em 1967 na Noruega), define um paradima de programação amplamente empregado no desenvolvimento de sistemas atuais. É uma tecnologia de programação baseda em elementos atômicos básicos chamados “objetos” que representam entes do mundo real, e está baseada em três pilares: *herança*, *polimorfismo* e *encapsulamento*.

**Herança** é um mecanismo pelo qual uma classe, denominada classe-filha ou classe derivada, herda propriedades e métodos de outra classe denominada classe-mãe, classe base ou superclasse. A herança permite reutilização de código e extensão de funcionalidade e podendo, ainda, definir *métodos* e *propriedades* próprias.

**Polimorfismo** é um mecanismo que permite que métodos com o mesmo nome possam realizar ações diferentes, dependendo do contexto ou do tipo de objeto. Por exemplo, um método chamado ***saudar()*** pode ter implementações diferentes em classes distintas, como 'Cliente' e 'Funcionário', adaptando-se ao comportamento específico de cada classe.

**Encapsulamento** é um mecanismo que oculta detalhes internos de implementação de uma classe, expondo apenas o necessário por meio de métodos públicos. Ele protege os dados de alterações externas indevidas permitindo, por exemplo, o uso de variáveis privadas que só podem ser acessadas ou modificadas por métodos específicos dentro da própria classe.

Assim, é lógico que a implementação de um programa com tecnologia orientada a objetos deve usar uma **OOL** (**O**bject-**O**riented **Language**) como por exemplo, Java, C++, Python e outras. Entretanto, para os programadores iniciantes, essas linguagens não são fáceis de aprender; assim, uma opção seria o bom e flexível Visualg; problema: Visualg não tem recursos de orintação a objetos. Entretanto, o porograma **“HerancaVisualg”** faz a simulação do conceito de herança usando o recurso de *Registro* (Tipo), que pode ser implementado pelos programadores iniiantes no aprendizado de OOP, baseando na classe primitiva “Veiculo”, e criando a classe derivada “Carro”. A saída deste programa, simulando a Herança em Visualg, é a seguinte:

**Marca: Toyota | Modelo: Corolla | Ano: 2022**

**| Portas: 4**

1. O programa define uma classe base **Veiculo** com os atributos ***marca***, ***modelo*** e ***ano***. Esses atributos são herdados pela classe derivada **Carro**.
2. A classe **Carro** adiciona um novo atributo, **portas**, que é específico para carros.
3. O método ***MostrarDados()*** exibe as informações herdadas da classe **Veiculo** (*marca*, *modelo* e *ano*).
4. O método ***MostrarDadosCarro()*** chama o método ***MostrarDados()*** da classe base para exibir os dados herdados e, em seguida, imprime o número de portas específico do carro.
5. No programa principal, o objeto **meuCarro** é criado e inicializado com os valores **"Toyota"**, **"Corolla"**, **2022** e **4** (para o número de portas). Quando o método ***MostrarDadosCarro()*** é chamado ele imprime todos os dados do carro.

Esta saída reflete o comportamento esperado do programa, demonstrando como a Herança permite reutilizar e estender funcionalidades em algoritmos estruturados como criados no Visualg

----------------------------------------------------------------------------------------------------------------------------------

**Algoritmo** **"HerancaVisualg"**

*//Definição da Classe Primitiva*

**Tipo**

**Veiculo** = **Registro**

marca: **caractere**

modelo: **caractere**

ano: **inteiro**

**FimRegistro**

*//Método para exibir informações do veículo*

**Funcao** **MostrarDados**(v: **Veiculo**)

**Escreval**("Marca: ", v.marca, " | Modelo: ", v.modelo, " | Ano: ", v.ano)

**FimFuncao**

*//----------------------------------------------------------------------------------*

*//Definição da Classe Derivada (simula a Herança)*

**Tipo**

**Carro** = **Registro**

herda **Veiculo** *//Herança da classe Veiculo*

portas: **inteiro**

**FimRegistro**

*//----------------------------------------------------------------------------------*

*// Método específico da classe Carro*

**Funcao** **MostrarDadosCarro**(c: **Carro**)

**MostrarDados**(c) *// Chama o método da classe base*

**Escreval**(" | Portas: ", c.portas)

**FimFuncao**

*//===================================================================================*

*//Programa Principal*

**var**

meuCarro: **Carro**

**Inicio**

*//Definição do objeto da classe Carro*

meuCarro.marca <- "Toyota"

meuCarro.modelo <- "Corolla"

meuCarro.ano <- 2022

meuCarro.portas <- 4

*//Exibe os dados do carro*

**MostrarDadosCarro**(meuCarro)

**FimAlgoritmo**