

Polimorfismo: C# e Python

Mário Leite

...

A Tecnologia de Orientação a Objeto, embora não seja perfeita e não muito trivial de aprender, é uma das mais utilizadas nos desenvolvimentos de sistemas. Criada na Noruega nos anos 1960, tornou-se bem popular no Brasil a partir dos anos 2000 com a popularização dos microcomputadores e alavancada pelos sistemas baseados em plataformas GUI, como o Windows. Mais precisamente pelo Delphi, que concorrente direto do Visual Basic que não implementava, totalmente, essa tecnologia. Por outro lado, o C++ que era (é) o representante mais completo da **OOP (Object-Oriented Programming)** sempre foi de aprendizado muito difícil e bastante complicado, o que é uma barreira para se tornar popular (pelo menos no Brasil).

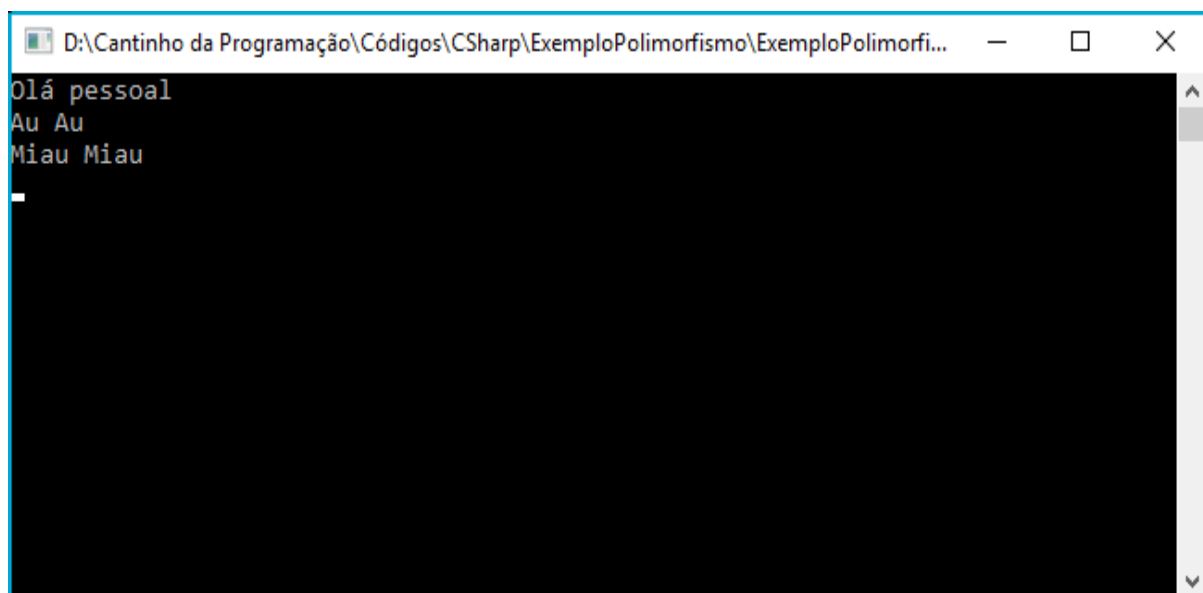
São quatro os pilares da OOP: **Abstração, Herança, Encapsulamento e Polimorfismo**. A Abstração é o mecanismo em que o programador deve se abster de todos os itens que não têm importância para o sistema; concentrando apenas no essencial. Herança, como o nome indica, permite que várias classes sejam criadas a partir de uma classe-mãe (também chamada de ‘classe ancestral’ ou ‘Primitiva’) herdando dessa classe TODOS os **atributos** (propriedades) e **métodos** (operações internas); e ainda podem criar seus próprios atributos e métodos. Encapsulamento é a capacidade de ocultar seus atributos e métodos, impedindo acessos externos e indevidos.

O termo “Polimorfismo”, etimologicamente, quer dizer “várias formas”; em particular no universo da OOP, é definido como sendo um código que possui “vários comportamentos” ou que produz “vários comportamentos”; em outras palavras, é um código que pode ser aplicado à várias classes de objetos. De maneira prática isto quer dizer que a operação em questão mantém seu comportamento transparente para quaisquer tipos de argumentos; isto é, a mesma mensagem é enviada a objetos de classes distintas e eles poderão reagir de maneiras diferentes. Um método polimórfico é aquele que pode ser aplicado à várias classes de objetos sem que haja qualquer inconveniente. É o caso por exemplo, do método **Clear** em Delphi®, que pode ser aplicado tanto à classe **TEdit** como à classe **TListBox**; nas duas situações o conteúdo desses objetos é “limpo”, mesmo pertencendo, ambos, à classes distintas. Um exemplo bem didático para o polimorfismo é dado por um simples moedor de carne. Esse equipamento tem a função de moer carne, produzindo carne moída para fazer bolinhos. Desse modo, não importa o tipo (classe) de carne alimentada; o resultado será sempre carne moída, não importa se de gado, de frango ou de qualquer outro tipo. As restrições impostas pelo processo estão no próprio objeto, definidas pelo seu fabricante e não pelo usuário do produto.

O programa “**ExemploPolimorfismo**”, implementado em C# e em Python, mostra um exemplo bem simples de Polimorfismo com três classes (ClsCachorro, ClsGato e ClsHumano) derivadas de uma classe primitiva: ClsAnimal. Assim, além do Polimorfismo o programa implementa, também, o mecanismo da Herança. A **figura 1** mostra a saída do programa em C# e a **figura 2** a saída em Python

Nota: Para saber mais sobre OOP acesse o *link* abaixo:

http://www2.dca.fee.unicamp.br/pesquisa/jornalLCA/centro.php?sessao_id=&login=&estilo=&conteudo_id=52

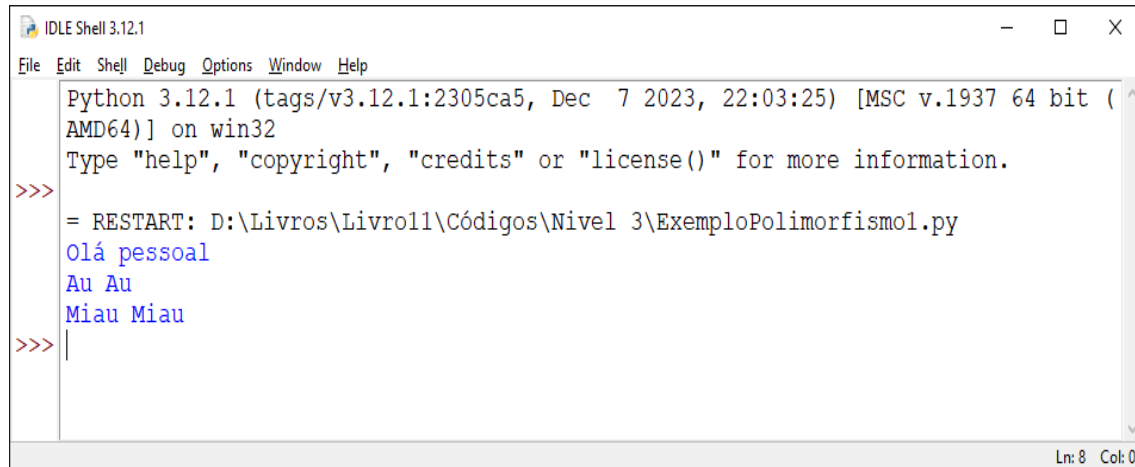


A screenshot of a console window titled "D:\Cantinho da Programação\Códigos\CSharp\ExemploPolimorfismo\ExemploPolimorfi...". The window has a black background with white text. The output consists of three lines: "Olá pessoal", "Au Au", and "Miau Miau". A white cursor is visible on the line following "Miau Miau".

```
Olá pessoal
Au Au
Miau Miau

```

Figura 1 - Saída do programa em C#



A screenshot of an IDLE Shell window titled "IDLE Shell 3.12.1". The window has a white background with black text. The output consists of three lines: "Olá pessoal", "Au Au", and "Miau Miau". A red prompt ">>>" is visible on the line following "Miau Miau". The status bar at the bottom right shows "Ln: 8 Col: 0".

```
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec 7 2023, 22:03:25) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: D:\Livros\Livro11\Códigos\Nivel 3\ExemploPolimorfismo1.py
Olá pessoal
Au Au
Miau Miau
>>>

```

Figura 2 - Saída do programa em Python

```

using System;

namespace ExemploPolimorfismo
{
    // Cria a classe ancestral (primitiva)
    public class ClsAnimal
    {
        public virtual string Comunicar()
        {
            return "";
        }
    }

    // Cria a primeira classe derivada
    public class ClsHumano : ClsAnimal
    {
        public override string Comunicar()
        {
            return "Olá pessoal";
        }
    }

    // Cria a segunda classe derivada
    public class ClsCachorro : ClsAnimal
    {
        public override string Comunicar()
        {
            return "Au Au";
        }
    }

    // Cria a terceira classe derivada
    public class ClsGato : ClsAnimal
    {
        public override string Comunicar()
        {
            return "Miau Miau";
        }
    }

    // Programa principal
    class Program
    {
        static void FazerSom(ClsAnimal animal)
        {
            Console.WriteLine(animal.Comunicar());
        }

        static void Main()
        {
            ClsAnimal animal1 = new ClsHumano();
            ClsAnimal animal2 = new ClsCachorro();
            ClsAnimal animal3 = new ClsGato();

            FazerSom(animal1); // Saída da comunicação do humano
            FazerSom(animal2); // Saída da comunicação do cachorro
            FazerSom(animal3); // Saída da comunicação do gato

            Console.ReadKey();
        }
    }
}
// Fim da aplicação "ExemploPolimorfismo"

```

```
'''
ExemploPolimorfismo.py
-----

Exemplo simples que demonstra o mecanismo de Polimorfismo.
Usa um mesmo método para ações diferentes.
-----
'''

#Cria a classe ancestral (primitiva)
class ClsAnimal:
    def Comunicar(self):
        pass

#-----
#Cria a primeira classe derivada
class ClsHumano(ClsAnimal):
    def Comunicar(self):
        return "Olá pessoal"

#-----
#Cria a primeira classe derivada
class ClsCachoro(ClsAnimal):
    def Comunicar(self):
        return "Au Au"

#-----
#Cria a segunda classe derivada
class ClsGato(ClsAnimal):
    def Comunicar(self):
        return "Miau Miau"

#Método que recebe "ClsAnimal" e chama o método comunicar
def FazerSom(ClsAnimal):
    print(ClsAnimal.Comunicar())

#=====
#Programa principal
#Criar instâncias das classes derivadas
ClsAnimal1 = ClsHumano()
ClsAnimal2 = ClsCachoro()
ClsAnimal3 = ClsGato()

#Chama o mesmo método com diferentes objetos
FazerSom(ClsAnimal1) #saída da comunicação do humano
FazerSom(ClsAnimal2) #saída da comunicação de cachorro
FazerSom(ClsAnimal3) #saída da comunicação de gato
#Fim do programa "ExemploPolimorfismo" -----
```