# **Aplicando os Pilares da Programação Orientada a Objetos em Python (POO) - Exemplo de Animais**

#### **Objetivo:**

Neste exercício, criamos um sistema simples de classes de animais para demonstrar os quatro pilares fundamentais da Programação Orientada a Objetos (POO): Encapsulamento, Abstração, Herança e Polimorfismo

#### **Pilares Fundamentais da POO:**

Os quatro pilares da POO:

1. **Encapsulamento**: Consiste em ocultar os detalhes de implementação de uma classe, expondo apenas o que é necessário para o uso externo.
2. **Abstração**: Permite focar nos aspectos essenciais de um objeto, ocultando detalhes irrelevantes.
3. **Herança**: Permite criar novas classes a partir de classes existentes, herdando seus atributos e métodos.
4. **Polimorfismo**: Permite que objetos de classes diferentes possam ser tratados de maneira uniforme.

Agora, vamos aplicar esses pilares no nosso exercício.

**1. Encapsulamento**:

O encapsulamento é aplicado na classe Animal:

* O atributo \_\_nome é privado, começando com dois underscores.
* O acesso ao nome é controlado pelo método getter get\_nome().
* Exemplo: self.\_\_nome = nome no construtor e return self.\_\_nome no getter.

**2. Abstração**:

A abstração é implementada através da classe abstrata Animal:

* Animal é uma classe abstrata que não pode ser instanciada diretamente.
* Ela contém um método abstrato fazer\_som().
* Isso permite definir uma interface comum para todos os animais, sem especificar como cada animal faz som.

**3. Herança**:

A herança é demonstrada nas classes Cachorro e Gato:

* Ambas as classes herdam da classe base Animal.
* Elas herdam o atributo \_\_nome e o método get\_nome().
* Cada subclasse adiciona seus próprios atributos específicos (raca para Cachorro e cor para Gato).

**4. Polimorfismo**:

O polimorfismo é aplicado através do método fazer\_som():

* Cada subclasse (Cachorro e Gato) implementa seu próprio método fazer\_som().
* Isso permite que diferentes objetos respondam ao mesmo método de maneiras distintas.
* Na função emitir\_som(), podemos chamar fazer\_som() em qualquer objeto Animal, independentemente de ser um cachorro ou um gato.

#### **Aplicação Prática**

1. **Encapsulamento**:
   * Protege o nome do animal de acesso direto (self.\_\_nome).
   * Fornece um método controlado para acessar o nome (get\_nome()).
2. **Abstração**:
   * A classe Animal define a estrutura básica para todos os tipos de animais.
   * O método abstrato fazer\_som() estabelece um contrato que todas as subclasses devem seguir.
3. **Herança**:
   * Cachorro e Gato herdam características comuns de Animal.
   * Cada subclasse adiciona atributos específicos (raca e cor).
4. **Polimorfismo**:
   * Permite que diferentes animais "façam som" de maneiras diferentes.
   * A função emitir\_som() pode trabalhar com qualquer objeto que seja um Animal.

## Conclusão

Ao aplicar esses pilares da POO no exercício de animais, criamos um sistema flexível e extensível. O código está organizado de forma que novos tipos de animais podem ser adicionados facilmente, apenas criando novas subclasses de Animal. O encapsulamento protege os dados internos, a abstração fornece uma interface comum, a herança permite a reutilização de código, e o polimorfismo permite tratar diferentes tipos de animais de maneira uniforme.