POO II

Letícia Vidal

Índice

O que é Abstração?

No mundo real, **abstração** significa focar nos aspectos essenciais de algo e ignorar os detalhes complexos de como funciona por baixo dos panos.

Abstração na Programação Orientada a Objetos (POO)

Na programação, a abstração é um dos 4 pilares da POO (junto com Encapsulamento, Herança e Polimorfismo). Ela serve para:

- **Ocultar a complexidade:** Mostrar apenas o que o objeto faz, não como ele faz.
- **Criar "contratos" (Templates):** Obrigar que classes filhas implementem certos métodos.
- **Evitar a criação de objetos incompletos:** Classes abstratas servem como base e não podem ser instanciadas (você não pode criar um objeto direto delas).

Como funciona no Python? (O módulo abc)

Diferente de linguagens como Java ou C#, o Python não tem a palavra-chave `abstract` embutida nativamente na sintaxe básica. Para usar abstração real, importamos o módulo embutido chamado **`abc`** (Abstract Base Classes).

Para criar uma classe abstrata no Python, você precisa:

1. Fazer sua classe herdar de `ABC`.
2. Usar o decorador `@abstractmethod` nos métodos que devem ser obrigatórios para as classes filhas.

Exemplo Prático: Formas Geométricas

Imagine que estamos criando um sistema de desenho. Sabemos que toda forma geométrica tem uma área, mas o cálculo da área de um Quadrado é totalmente diferente do cálculo de um Círculo.

Criaremos uma classe abstrata `Forma` que obriga todas as formas a terem um método `calcular_area()`.

```
from abc import ABC, abstractmethod
import math

# 1. Criando a Classe Abstrata
class Forma(ABC):

    @abstractmethod
    def calcular_area(self):
        # Este método não tem implementação na classe base.
        # Ele serve apenas como uma "regra" para as filhas.
        pass

    def descricao(self):
        # Uma classe abstrata também pode ter métodos normais (concretos)
        return "Eu sou uma forma geométrica."

# 2. Criando Classes Concretas (Filhas)
class Quadrado(Forma):
    def __init__(self, lado):
        self.lado = lado

    # Se não implementarmos 'calcular_area', o Python dará erro!
    def calcular_area(self):
        return self.lado * self.lado

class Circulo(Forma):
    def __init__(self, raio):
        self.raio = raio

    def calcular_area(self):
        return math.pi * (self.raio ** 2)
```

Exemplo Prático: Formas Geométricas

```
# --- Testando o Código ---

# forma = Forma()  # ERRO! Não podemos instanciar uma classe abstrata.

quadrado = Quadrado(4)
circulo = Circulo(3)

print(quadrado.descricao())                 # Saída: Eu sou uma forma geométrica.
print(f"Área do quadrado: {quadrado.calcular_area()}") # Saída: Área do quadrado: 16

print(circulo.descricao())                  # Saída: Eu sou uma forma geométrica.
print(f"Área do círculo: {circulo.calcular_area():.2f}") # Saída: Área do círculo: 28.27
```

O que acontece se eu esquecer de implementar o método?

Se você criar uma classe filha (ex: `Triangulo`) e esquecer de criar o método `calcular_area()`, o Python **não deixará você criar o objeto**. Ele lançará um erro logo no momento em que você tentar instanciar: `TypeError: Can't instantiate abstract class Triangulo with abstract method calcular_area`.

Resumo e Benefícios

Benefício	Descrição
Padronização	Garante que diferentes desenvolvedores criem classes com a mesma estrutura obrigatória.
Segurança	Impede que objetos "genéricos" ou incompletos sejam criados no sistema.
Manutenção	Facilita a leitura do código, pois a classe base funciona como um índice do que a classe faz.

Exercício de Fixação

Desafio: 1. Crie uma classe abstrata chamada `ContaBancaria` utilizando o módulo `abc`. 2. Ela deve ter um atributo `saldo` e dois métodos abstratos: `depositar(valor)` e `sacar(valor)`. 3. Crie duas classes filhas: `ContaCorrente` e `ContaPoupanca` que herdem de `ContaBancaria`. 4. A `ContaCorrente` tem uma taxa de R$ 2,00 por saque. A `ContaPoupanca` não tem taxa, mas não permite que o saldo fique negativo.

resolução