Herança Múltipla

Mário Leite

Os pilares da Tecnologia de Orientação a Objetos são baseados em quatro mecanismos: Abstração, Encapsulamento, Polimorfismo e Herança. Este último é mais fácil de entender e o mais importante, pois permite fazer com que uma classe (coleção de características comportamentos de um tipo de objeto) herde todas essas características de outra classe. A classe que doa é chamada "classe mãe" e a que herda "classe filha". Existem dois tipos de herança: Simples e Múltipla.

A **Herança Múltipla** é um conceito da Programação Orientada a Objetos (POO) no qual uma classe pode herdar características (atributos e métodos) de mais de uma classe base ao mesmo tempo. Em vez de ter apenas um "pai", a classe derivada passa a ter vários ancestrais, combinando comportamentos distintos em uma única estrutura.

Na herança simples, uma classe filha herda de **apenas uma** classe base; na **herança múltipla**, a classe filha herda de **duas ou mais** classes bases. Um exemplo conceitual (mundo real); imagine as classes:

- **Impressora**
- **Scanner**
- **Multifuncional**

Uma **multifuncional** é **impressora** *e também* **Scanner**, pois herda o comportamentos de ambos.

Por que usar Herança Múltipla?

A herança múltipla é útil quando:

- Um objeto **naturalmente combina responsabilidades distintas**.
- Há **reuso real de comportamento**, não apenas de código.
- As classes base representam **papéis independentes**.

Ela permite:

- Evitar duplicação de código.
- Modelar sistemas mais próximos da realidade.
- Combinar funcionalidades de forma declarativa.

Observe o exemplo conceitual abaixo...

```
Classe Impressora
   Método Imprimir()

Classe Scanner
   Método Digitalizar()

Classe Multifuncional herda Impressora, Scanner
```

```
Assim, classe Multifuncional passa a ter:

  Imprimir() (da Impressora)
  Digitalizar() (do Scanner)
```

Sem precisar reimplementar nada.
Os dois programas abaixo mostram exemplos práticos de aplicação da Herança Múltipla, com as respectivas saídas.

```
'''
ExemploHerancaMultipla1.py
-------------------------------------------------------------------------------
Cria uma classe "ClsFilho" herdando das classes: "ClsPai" e "ClsMae"
(simultaneamente) e mostra como a classe derivada pode acessa as classes
ancestrais, mostrando o mecanismo de herança múltipla.
-------------------------------------------------------------------------------
'''
class ClsPai:
    def __init__(self):
        self.caracteristica_pai = "Alto, negro, cabelos crespos"

    def ExecutarMetodoPai(self):
        print("Método do pai:", "TrabalharFora")
#-------------------------------------------------------------------------------
class ClsMae:
    def __init__(self):
        self.caracteristica_mae = "Baixa, branca, cabelos lisos"
    def ExecutarMetodoMae(self):
        print("Método da mãe:", "TrabalharEmCasa")
#-------------------------------------------------------------------------------
class ClsFilho(ClsPai, ClsMae):
    def __init__(self):
        #Chama os construtores das duas classes-base
        ClsPai.__init__(self)
        ClsMae.__init__(self)
        self.caracteristica_filho = "Característica do filho"
    def ExecutarMetodoFilho(self):
        print("Método do filho:", "EstudarPython")
#===============================================================================
#Programa principal
Filhote = ClsFilho()  #cria uma instância de "ClsFilho"
#Acessa as características e métodos de "Filhote"
print("Característica do pai:",   Filhote.caracteristica_pai)
print("Característica da mãe:",   Filhote.caracteristica_mae)
print("Característica do filho:", Filhote.caracteristica_filho)
Filhote.ExecutarMetodoPai()
Filhote.ExecutarMetodoMae()
Filhote.ExecutarMetodoFilho()
#Fim do programa "ExemploHerancaMultipla1" ---------------------------------
```

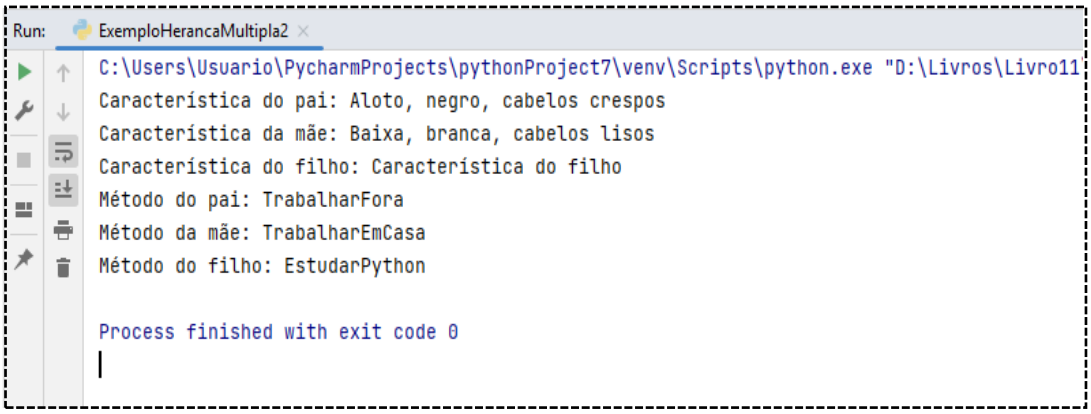

Figura 1 - Saída do programa "ExemploHerancaMultipla1"

```
'''
ExemploHerancaMultipla2.py
-----------------------------------------------------------------------------------------
Faz herança múltipla com um exemplo de aplicação financeira, onde a classe "ClsConta"
possui métodos para manipular o saldo de uma conta bancária, enquanto a classe
"ClsRendimento" possui um método para calcular o rendimento com base em uma taxa. A
classe  classe "ClsInvestimento" herda tanto da classe "ClsConta" quanto da classe
"ClsRendimento".
-----------------------------------------------------------------------------------------
'''
class ClsConta:
    def __init__(self, saldo_conta):
        self.saldo_conta = saldo_conta

    def Depositar(self, valor):
        self.saldo_conta += valor

    def Sacar(self, valor):
        if self.saldo_conta >= valor:
            self.saldo_conta -= valor
            return True
        else:
            return False
#-----------------------------------------------------------------------------------------
class ClsRendimento:
    def CalcularRendimento(self, taxa):
        self.saldo_conta += self.saldo_conta * (taxa / 100)
#-----------------------------------------------------------------------------------------
class ClsInvestimento(ClsConta, ClsRendimento):
    def __init__(self, saldo_conta):
        super().__init__(saldo_conta)
#=========================================================================================
#Programa principal
invInicial = 0
while(invInicial<1000):
    invInicial = abs(float(input("Entre com o investimento inicial: ")))

taxa = 0
while(taxa<=0):
    taxa = abs(float(input("Entre com a taxa de juros [decimal]: ")))
ContaInv = ClsInvestimento(invInicial)
print("Saldo inicial:", ContaInv.saldo_conta)  #cria instância para manipular valores
deposito = 0
while(deposito<100):
    deposito = abs(float(input("Entre com o valor do depósito [min: 100]: ")))
ContaInv.Depositar(deposito)
print("Saldo após depósito:", ContaInv.saldo_conta)
ContaInv.CalcularRendimento(taxa)
print("Saldo após rendimento de 1.5%:", ContaInv.saldo_conta)
#Fim do programa "ExemploHerancaMultipla2" ----------------------------------------------
```

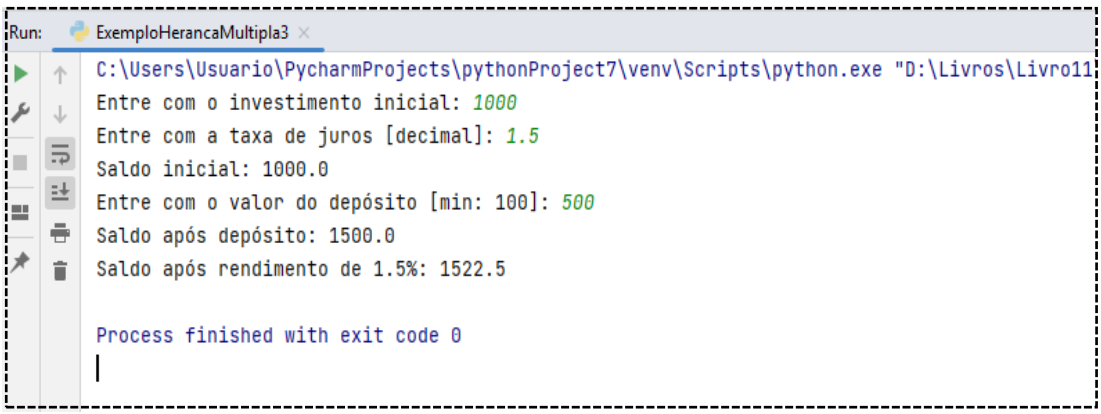

Figura 2 - Saída do programa "ExemploHerancaMultipla2"