

UNIVERSIDADE FEDERAL DO AMAZONAS – UFAM
INSTITUTO DE COMPUTAÇÃO - ICOMP

USANDO CONCEITOS DE POO NO CONTEXTO DE TDD

ANDRÉ VIEIRA DOS SANTOS
KELEN RODRIGUES DA SILVA LIMA
RAUL CARDOSO BATALHA

MANAUS – AM

2022

INTRODUÇÃO

Conceito bastante utilizado em Python para definir herança é através do paradigma da orientação à objetos OO, onde se determina que a classe principal indica a herança dos atributos e métodos de uma outra classe e, assim, evitando que ocorra muita repetição de código.

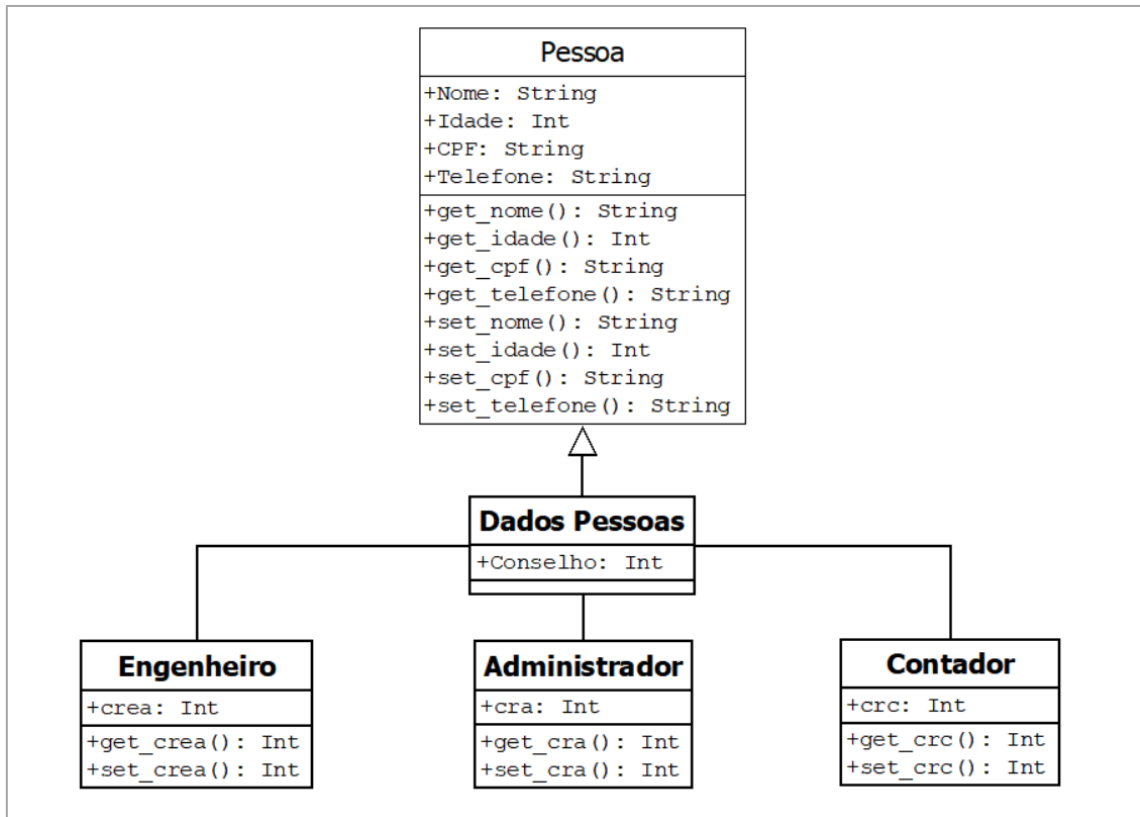
Mediante ao conceito principal de herança, propomos uma exemplificação simples e de fácil entendimento, apresentando o exemplo de herança através da Classe principal Pessoa.

Assim, a Classe principal Pessoa determina os atributos e métodos principais, que podem ser herdados para funções profissionais.

Utilizamos as classes Engenheiro, Contador e Administrador, para herdar os atributos e métodos da classe principal.

A HIERARQUIA

O diagrama a seguir demonstra o exemplo utilizado:



1. Herança

A classe principal utilizada para indicar os atributos e métodos a serem utilizados em outra classe foi nomeada como classe pessoa:

```
from abc import ABC, abstractclassmethod
from dadosPessoas import DadosPessoas

class Pessoa(ABC):

    def __init__(self, nome, idade, cpf, telefone):
        self.__nome = nome
        self.__idade = idade
        self.__cpf = cpf
        self.__telefone = telefone

    def get_conselho(self):
        pass

    def get_nome(self):
        pass

    def get_idade(self):
        pass

    def get_cpf(self):
        pass

    def get_telefone(self):
        pass

    def set_conselho(self):
        pass

    def set_nome(self, nome):
        pass

    def set_idade(self, idade):
        pass

    def set_cpf(self, cpf):
        pass

    def set_telefone(self, telefone):
        pass
```

2. Polimorfismo

Polimorfismo pôde ser representado através da subclasse DadosPessoa, onde apesar de conter atributos semelhantes ao da Classe Principal, Pessoa, ela especificou o atributo que pode ser determinante para identificar o conselho para determinada pessoa.

```
class DadosPessoas():

    def __init__(self, conselho, nome, idade, cpf, telefone):
        self.__telefone = telefone

    def get_conselho(self):
        return self.__conselho

    def get_nome(self):
        return self.__nome

    def get_idade(self):
        return self.__idade

    def get_cpf(self):
        return self.__cpf

    def get_telefone(self):
        return self.__telefone

    def set_conselho(self):
        return self.__conselho

    def set_nome(self, nome):
        self.__nome = nome

    def set_idade(self, idade):
        self.__idade = idade

    def set_cpf(self, cpf):
        self.__cpf = cpf

    def set_telefone(self, telefone):
        self.__telefone = telefone
```

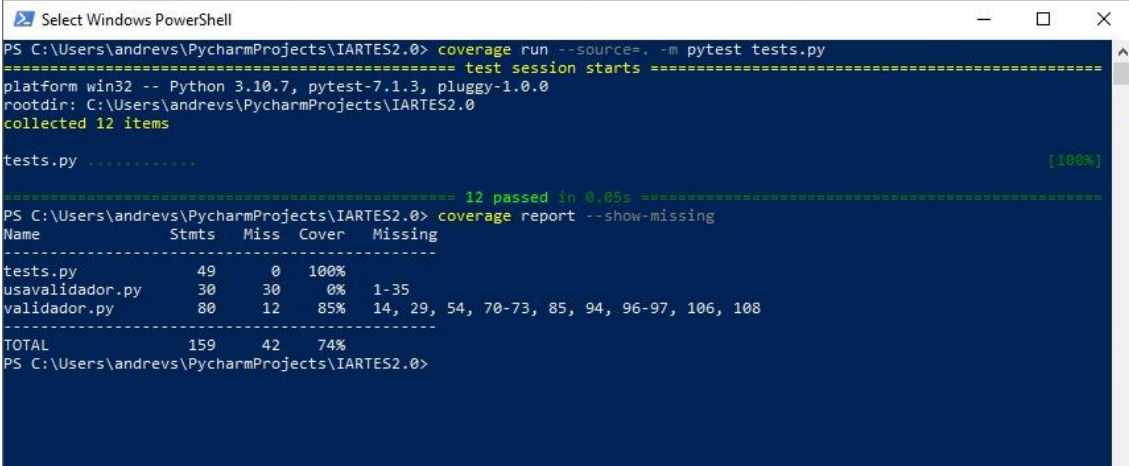
3. Encapsulamento

O encapsulamento utilizado foi na definição dos atributos e métodos iniciados por dois sublinhados tornando-os como privados.

```
class Pessoa(ABC):  
  
    def __init__(self, nome, idade, cpf, telefone):  
        self.__nome = nome  
        self.__idade = idade  
        self.__cpf = cpf  
        self.__telefone = telefone
```

4. Relatório de cobertura do código

A cobertura do código alcançada ao final dos testes foi de 74%:



```
Select Windows PowerShell  
PS C:\Users\andrevs\PycharmProjects\IARTES2.0> coverage run --source=. -m pytest tests.py  
===== test session starts =====  
platform win32 -- Python 3.10.7, pytest-7.1.3, pluggy-1.0.0  
rootdir: C:\Users\andrevs\PycharmProjects\IARTES2.0  
collected 12 items  
  
tests.py ..... [100%]  
  
===== 12 passed in 0.05s =====  
PS C:\Users\andrevs\PycharmProjects\IARTES2.0> coverage report --show-missing  
Name           Stmts  Miss  Cover   Missing  
-----  
tests.py         49      0   100%  
usavalidador.py   30     30      0%    1-35  
validador.py      80     12    85%    14, 29, 54, 70-73, 85, 94, 96-97, 106, 108  
-----  
TOTAL             159     42    74%
```

CONCLUSÃO

A herança em Python OO, acontece quando duas classes são próximas, e possuem características bilaterais, porém não são 100% iguais, podendo existir atributos específicos para diferenciá-las uma da outra. Portanto, ao invés de ser necessário a definição de todo o código novamente, pode se especificar definindo pontos da classe derivada, aquela classe que herdou os atributos e métodos da classe principal.

Em todo desenvolvimento da atividade podemos identificar que a herança é a parte importante da OO, pois ela permite a reutilização de código existente e facilita o projeto, já que é possível organizar os códigos separadamente sem que precisem ficar todos dentro de um único arquivo, além de permitir o aproveitamento, deixando o programa de forma mais enxuta e organizado.