Relatório

Conceitos de POO no contexto de TDD

Equipe:
Ana Paula Cardoso
Marlon Griego
Pablo Diego
Cleicy Priscilla Aragão dos Santos

1. Introdução.

Para esse relatório final usamos o conceito de criação de contas bancárias e movimentação.

Para as **Classe Raiz**: Classe Movimento

Para **Classe abstrata**: Classe Pessoa

2. Herança

A classe **pessoa_juridica** e a classe **pessoa_fisica** herdam seus atributos da classe pessoa.py

```python
class ContaComum(object):

    global lista_contas
    lista_contas = []

    # Construtor
    def __init__(self, nro_conta=0, dt_abertura='', dt_encerramento='', saldo=0.0):
        self.nro_conta = nro_conta
        self.dt_abertura = dt_abertura
        self.dt_encerramento = dt_encerramento
        self.saldo = saldo
```

```python
from itertools import cycle

from pessoa import Pessoa


class PessoaJuridica(Pessoa):

 #Construtor
    def __init__(self, cnpj='', razao_social='', nome_fantasia=''):
        self.cnpj = cnpj
        self.razao_social = razao_social
        self.nome_fantasia = nome_fantasia
```

```python
import re

from pessoa import Pessoa


class PessoaFisica(Pessoa):

 #Construtor
    def __init__(self, cpf='', rg='', dependentes=False):
        self.cpf = cpf
        self.rg = rg
        self.dependentes = dependentes

```

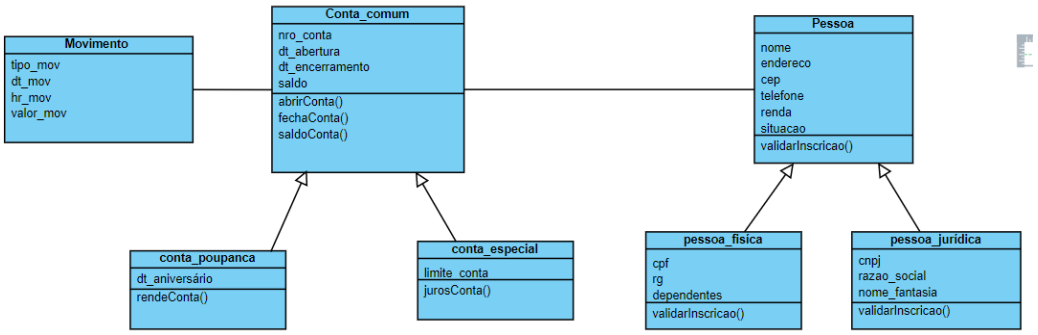


Figura-1 Diagrama de Classe

3.Polimorfismo

Para validar a inscrição dentro da classe **Pessoa** foi criado o método **validarInscrição,** onde a implementação varia **na classe pessoa_juridica** e **pessoa_fisica.**

```python
#Metodos
#Polimorfismo, utizando o metodo da classe Pessoa
    def validarInscricao(self, cnpj):
        LENGTH_CNPJ = 14
        if len(cnpj) != LENGTH_CNPJ:
            print("teste1", cnpj)
            return False

        if cnpj in (c * LENGTH_CNPJ for c in "1234567890"):
            print("teste2", cnpj)
            return False

        cnpj_r = cnpj[::-1]
        for i in range(2, 0, -1):
            cnpj_enum = zip(cycle(range(2, 10)), cnpj_r[i:])
            dv = sum(map(lambda x: int(x[1]) * x[0], cnpj_enum)) * 10 % 11
            if cnpj_r[i - 1:i] != str(dv % 10):
                return False

        print("testeTrue", cnpj)
        return True
```

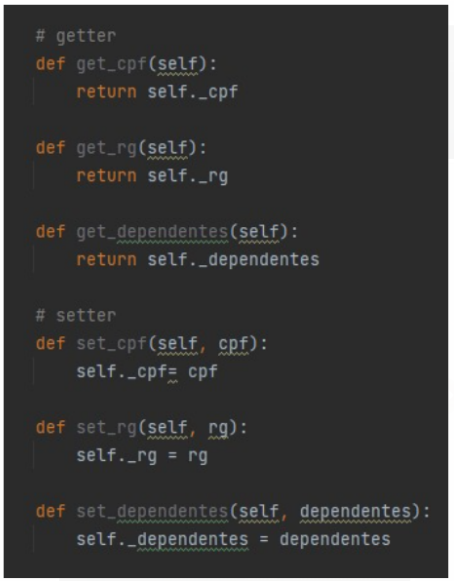
```python
#Metodos
#Polimorfismo, utizando o metodo da classe Pessoa
def validarInscricao(self,cpf):
    # Verificar a formatação do CPF
    if not (re.match(r'\d{3}\.\d{3}\.\d{3}-\d{2}', cpf)):
        return False
    else:
        # Obter os numeros, sem digitos
        cpf_numeros = [int(digito) for digito in cpf if digito.isdigit()]
        # Verificar se o CPF tem 11 digitos ou se todos os numeros sao iguais
        if (len(cpf_numeros) != 11 or len(set(cpf_numeros)) == 1):
            return False

        ## Validar o primeiro digito verificador:
        soma = sum(a * b for a, b in zip(cpf_numeros[0:9], range(10, 1, -1)))
        digito_esperado = (soma * 10 % 11) % 10
        if cpf_numeros[9] != digito_esperado:
            return False

        # Validar o segundo digito verificador:
        soma = sum(a * b for a, b in zip(cpf_numeros[0:10], range(11, 1, -1)))
        digito_esperado = (soma * 10 % 11) % 10
        if cpf_numeros[10] != digito_esperado:
            return False
```

```python
def validarInscricao(self,dados):
    pass
```

4.Encapsulamento.

Usamos encapsulamento para proteger os dados da conta na classe **conta_comum** e proteger os dados da pessoa classe **pessoa_fisica**.

```python
# getter
def get_nro_conta(self):
    return self._nro_conta

def get_dt_abertura(self):
    return self._dt_abertura

def get_dt_encerramento(self):
    return self._dt_encerramento

def get_saldo(self):
    return self._saldo

# setter
def set_nro_conta(self, nro_conta):
    self._nro_conta = nro_conta

def set_dt_abertura(self, dt_abertura):
    self._dt_abertura = dt_abertura

def set_dt_encerramento(self, dt_encerramento):
    self._dt_encerramento = dt_encerramento

def set_saldo(self, saldo):
    self._saldo = saldo

def abrirConta(self, nro_conta):
    if nro_conta in lista_contas:
        return False
    else:
        lista_contas.append(nro_conta)
        return True
```



```python
# getter
def get_cpf(self):
    return self._cpf

def get_rg(self):
    return self._rg

def get_dependentes(self):
    return self._dependentes

# setter
def set_cpf(self, cpf):
    self._cpf= cpf

def set_rg(self, rg):
    self._rg = rg

def set_dependentes(self, dependentes):
    self._dependentes = dependentes
```

5. Relatório de cobertura do código.

5.1 Cobertura inicial

```
Terminal:  Local  +
(venv) PS C:\Users\marlo\PycharmProjects\pythonprojetofinal> coverage report --show-missing
Name                 Stmts   Miss  Cover   Missing
--------------------------------------------------
conta_comum.py          37     23    38%   8-11, 16, 19, 22, 25, 29, 32, 35, 38, 41-45, 48-52, 55-58
conta_especial.py       15     11    27%   8-19, 24-30
conta_poupanca.py       12      8    33%   8-19, 23-26
movimento.py            55     43    22%   5-8, 13, 16, 19, 22-23, 27, 30, 33, 36, 39, 42-76
pessoa.py               34     32     6%   5-50
pessoa_fisica.py        34     29    15%   10-32, 38-59
pessoa_juridica.py      35     11    69%   16, 19, 22, 26, 29, 32, 39-40, 43-44, 51
testes.py               21      0   100%
--------------------------------------------------
TOTAL                  243    157    35%
(venv) PS C:\Users\marlo\PycharmProjects\pythonprojetofinal>
```

```
(venv) PS C:\Users\marlo\PycharmProjects\pythonprojetofinal> coverage report --show-missing
Name                 Stmts   Miss  Cover   Missing
--------------------------------------------------
conta_comum.py          37     23    38%   8-11, 16, 19, 22, 25, 29, 32, 35, 38, 41-45, 48-52, 55-58
conta_especial.py       15     11    27%   8-19, 24-30
conta_poupanca.py       12      8    33%   8-19, 23-26
movimento.py            55     43    22%   5-8, 13, 16, 19, 22-23, 27, 30, 33, 36, 39, 42-76
pessoa.py               34     32     6%   5-50
pessoa_fisica.py        34      7    79%   16, 19, 22, 26, 29, 32, 59
pessoa_juridica.py      35     11    69%   16, 19, 22, 26, 29, 32, 39-40, 43-44, 51
testes.py               39      0   100%
--------------------------------------------------
TOTAL                  261    135    48%
(venv) PS C:\Users\marlo\PycharmProjects\pythonprojetofinal>
```

5.2 Cobertura Final

```
Terminal:  Local  +
(venv) PS C:\Users\marlo\PycharmProjects\pythonprojetofinal> coverage report --show-missing
Name                 Stmts   Miss  Cover   Missing
--------------------------------------------------
conta_comum.py          37      8    78%   16, 19, 22, 25, 29, 32, 35, 38
conta_especial.py       15      2    87%   13, 19
conta_poupanca.py       12      2    83%   13, 19
movimento.py            55     14    75%   13, 16, 19, 22-23, 27, 30, 33, 36, 59-60, 66, 68-70
pessoa.py               34     32     6%   5-50
pessoa_fisica.py        34      6    82%   16, 19, 22, 26, 29, 32
pessoa_juridica.py      35      6    83%   16, 19, 22, 26, 29, 32
testes.py               91      0   100%
--------------------------------------------------
TOTAL                  313     70    78%
(venv) PS C:\Users\marlo\PycharmProjects\pythonprojetofinal>
```

6. Conclusões.

Criamos classe para simular a criação de conta bancária e movimentação, também foi desenvolvido cálculo de rendimento dessas contas.
Durante o desenvolvimento, tivemos dificuldade de encontrar uma classe usada no cotidiano, que atendesse todos os requisitos do trabalho final. Também tivemos algumas dificuldades para definição de métodos para algumas classes.