02 - Método com Exception

A seguir, vamos focar no calcular\_bonus, um dos métodos que a Dominique desenvolveu no início do projeto dela.

A Bytebank está adotando um plano para conferir um bônus salarial para alguns funcionários. Para receber esse bônus, 10% do salário da pessoa deve ser igual ou menor que R$1.000,00. Caso a pessoa se enquadre nessa regra, ela ganhará um bônus equivalente a 10% de seu salário.

A Dominique já criou o método calcular\_bonus, que retorna o valor do bônus a que o funcionário tem direito:

def calcular\_bonus(self):

valor = self.\_salario \* 0.1

if valor > 1000:

valor = 0

return valor

Temos uma variável chamada valor que inicialmente corresponde a 10% do salário do funcionário. Se essa variável for maior que 1000, atribuiremos a ela o valor 0. Do contrário, se a pessoa não se enquadrar nessa condição, manteremos o valor inicial. Ao final, retornamos valor.

Para verificar seu funcionamento, vamos ao arquivo main.py. Já transpomos o teste\_idade para o Pytest, então podemos apagá-lo. Em seguida, vamos instanciar uma funcionária chamada ana. Seu salário será R$1.000,00, ou seja, ela atende a condição para receber o bônus. Depois, vamos chamar o método calcular\_bonus e printar o resultado:

from codigo.bytebank import Funcionario

ana = Funcionario('Ana', '12/03/1997', 1000)

print(ana.calcular\_bonus())

Ao rodar o teste com "Ctrl + Shift + F10", o retorno será o valor do bônus salarial ao qual a Ana tem direito. No caso, R$100,00.

Vamos aumentar o salário da Ana, de modo que ela deixe de se enquadrar na regra:

from codigo.bytebank import Funcionario

ana = Funcionario('Ana', '12/03/1997', 100000000)

print(ana.calcular\_bonus())

Executando o teste, o retorno será 0. Segundo as regras da empresa, ela não se enquadra entre as pessoas que recebem o bônus, pois 10% de seu salário é maior que R$1.000,00.

Então, já entendemos o que o método calcular\_bonus faz e verificamos na prática seu funcionamento, porém ainda não existe nenhum teste associado a ele. Vamos desenvolvê-lo, a seguir. A partir da linha 34 do arquivo test\_bytebank.py, criaremos o teste test\_quando\_calcular\_bonus\_recebe\_1000\_deve\_retornar\_100. O raciocínio será semelhante aos anteriores que desenvolvemos, então basta seguir o mesmo modelo:

def test\_quando\_calcular\_bonus\_recebe\_1000\_deve\_retornar\_100(self):

entrada = 1000 # given

esperado = 100

funcionario\_teste = Funcionario('teste', '11/11/2000', entrada)

resultado = funcionario\_teste.calcular\_bonus() # when

assert resultado == esperado # then

A entrada será o salário de R$1.000,00 — este é o *given*. O resultado esperado, R$100,00. Em seguida, vamos instanciar o funcionario\_teste cujo salário é a entrada que declaramos no início da função. A variável resultado armazenará o valor obtido com a chamada do método calcular\_bonus — este é o *when*. Por fim, usamos o assert para comparar o resultado com o valor esperado — este é o *then*. O teste está pronto, vamos rodá-lo com o comando pytest -v.

Todos os testes passarão!

Mostramos o resultado para o chefe da Dominique e ele solicitoou uma modificação. Quando um funcionário não se enquadrar nas regras para receber o bônus, em vez de retornar 0, ele gostaria que retornássemos uma *exception*.

No Python, uma ***exception*** é um erro personalizado que podemos criar. Existem dois tipos de erro no Python: o erro de sintaxe e as *exceptions* (ou *runtime erros*). Para exemplificar os dois casos, vamos propositalmente causar alguns erros, no arquivo main.py.

De início, comentaremos o teste relativo ao método calcular\_bonus. Depois, na linha 9, vamos printar uma *string* qualquer e colocar um parêntese a mais no final:

from codigo.bytebank import Funcionario

'''

ana = Funcionario('Ana', '12/83/1997', 100000000)

print (ana.calcular\_bonus())

'''

print('evfvf'))

Ao rodar o teste com "Ctrl + Shift + F10", será indicado no terminal que há um **erro de sintaxe** (SyntaxError) no código, ou seja, cometemos um erro na própria linguagem.

Na sequência, vamos alterar a linha 9 para printar o resultado da divisão de 0 por 0, uma operação impossível:

from codigo.bytebank import Funcionario

'''

ana = Funcionario('Ana', '12/83/1997', 100000000)

print (ana.calcular\_bonus())

'''

print(0/0)

Ao executar o teste, teremos um erro diferente: ZeroDivisionError. Esse problema é um ***runtime error***, um erro na lógica do código. Trata-se de um erro de uma *exception*.

É possível criar *exceptions* personalizadas para situações que nós mesmos impomos. Trata-se de uma prática comum tanto no Python quanto em outras linguagens. Uma das vantagens de *exceptions* personalizadas é que podemos justificar por que não é possível fazer determinadas ações.

No arquivo bytebank.py, no método calcular\_bonus, quando a variável valor for maior que 1000, em vez de retornar o valor igual a 0, retornaremos uma *exception* personalizada com uma mensagem de erro:

def calcular\_bonus(self):

valor = self.\_salario \* 0.1

if valor > 1000:

raise Exception('O salário é muito alto para receber um bônus')

return valor

Em seguida, voltaremos ao main.py e apagaremos o exemplo da linha 9. Em seguida, vamos descomentar o teste da Ana e rodá-lo novamente:

from codigo.bytebank import Funcionario

ana = Funcionario('Ana', '12/03/1997', 100000000)

print(ana.calcular\_bonus())

Sabemos que 10% do salário da Ana é maior que R$1.000,00, por isso ela não tem direito ao bônus. Antes, o retorno era 0. A pedido do chefe da Dominique, fizemos algumas modificações e agora o retorno é uma exceção com a justificativa de que o salário é muito alto para receber um bônus. Para nos certificar de que os testes continuam funcionando, vamos executar pytest -v. O resultado indicará que todos continuam passando, sem problemas.