05 - Refatorando o código

Recapitulando o que fizemos até agora, nesta aula: chegou a demanda de uma nova funcionalidade, nós criamos os testes e, como esperado, eles falharam. Em seguida, desenvolvemos um código e fizemos os testes passarem. Ou seja, atendemos os requisitos mínimos necessários para que os testes fossem bem-sucedidos.

Agora, prestando atenção ao código que desenvolvemos, percebemos que ele não está condizendo com as boas práticas de desenvolvimento. O método decrescimo\_salario desempenha diversas funções, quando ideal é que cada método exerça apenas uma função.

Atualmente, verificamos se o funcionário é um sócio ou diretor e depois realizamos o decréscimo do salário. Seria uma boa prática termos dois métodos separados — um somente para identificar se o funcionário é um diretor ou sócio, e outro para fazer apenas o decréscimo (no caso, o decrescimo\_salario).

Com isso em mente, vamos refatorar esse código, dividindo-o em dois métodos distintos. Algumas linhas acima de decrescimo\_salario, criaremos o método \_eh\_socio. Por se tratar de um método que usaremos apenas dentro dessa classe e que não queremos que usuários tenham acesso, utilizamos o *underline* no início de seu nome para defini-lo como mais privado.

Em seguida, precisamos considerar a lógica que usamos em decrescimo\_salario e passá-la para \_eh\_socio. Basta copiarmos a lista sobrenomes e a condicional, pois sabemos que todo sócio ou diretor tem um salário de, pelo menos, R$100.000,00 e seu sobrenomes está nessa lista:

def \_eh\_socio(self):

sobrenomes = ['Bragança', 'Windsor', 'Bourbon', 'Yamato', 'Al Saud', 'Khan', 'Tudor', 'Ptolomeu']

if (self.\_salario >= 100000) and (self.sobrenome() in sobrenomes):

Se o funcionário for um sócio, vamos retornar True. Do contrário, retornaremos False:

def \_eh\_socio(self):

sobrenomes = ['Bragança', 'Windsor', 'Bourbon', 'Yamato', 'Al Saud', 'Khan', 'Tudor', 'Ptolomeu']

if (self.\_salario >= 100000) and (self.sobrenome() in sobrenomes):

return True

else:

return False

No método decrescimo\_salario, não precisaremos mais da lista de sobrenomes nem da condicional que criamos. Em vez disso, vamos chamar o método \_eh\_socio:

def decrescimo\_salario(self):

if self.\_eh\_socio():

decrescimo = self.\_salario \* 0.1

self.\_salario = self.\_salario - decrescimo

Vamos salvar as alterações. Depois abriremos o terminal e executaremos o pytest -v novamente. Descobriremos que todos os testes continuam passando, o que quer dizer que continuamos atendendo os requisitos mínimos dos nossos testes.

Ainda refletindo sobre esse código, vamos perceber que ainda há pontos de melhoria. A estrutura if/else não está escrita de uma maneira muito pythônica, é possível tornar esse trecho mais enxuto e legível:

def \_eh\_socio(self):

sobrenomes = ['Bragança', 'Windsor', 'Bourbon', 'Yamato', 'Al Saud', 'Khan', 'Tudor', 'Ptolomeu']

return (self.\_salario >= 100000) and (self.sobrenome() in sobrenomes)

Dessa forma, retornaremos o valor resultante dessas duas verificações. Caso ambas sejam verdadeiras, o funcionário é sócio ou diretor, e o retorno será True. Do contrário, teremos False.

Vamos rodar pytest -v para nos certificar de que tudo continua funcionando e constataremos que os testes ainda são bem-sucedidos. Com a refatoração, temos um código melhor organizado, mais legível e que atende às boas práticas de desenvolvimento.

Então, relembrando: havia uma nova funcionalidade para ser inserida no projeto, desenvolvemos os testes, eles falharam, então criamos a implementação do código, os testes passaram e, por fim, focamos na refatoração. Percebemos que um método desempenhava diversas funções, por isso o dividimos em dois e melhoramos sua estrutura, tornando-o mais legível e pythônico.

Nosso próximo passo será focar no método calcular\_bonus, que define se um funcionário tem direito a um bônus. No próximo video, vamos analisá-lo e decidir se ele respeita as regras de negócio, além de criar um teste para ele.