07

Faça o que eu fiz na aula

Enquanto escrevíamos os testes, vimos que havia algo de estranho com o design das classes, mas o que está estranho?

Para adicionar um leilão na lista, primeiro pedimos a lista de lances e depois utilizamos o método `append` da lista:

```
leilao.lances.append(lances)
```

Qual o problema com essa abordagem? O que acontece se trocarmos a implementação da classe `Leilao` ? O código provavelmente quebrará.

Ou seja, estamos muito acoplados com esse código da classe `Leilao` . O que fazer então? Uma forma é isolar a adição de um novo leilão na lista. Podemos fazer isso criando um novo método:

```
def propoe(self, lance: Lance):
        self.__lances.append(lance)
```

O método `propoe` fica sendo o responsável por adicionar um lance a lista. Dessa forma, escondemos, isto é, encapsulamos a implementação da classe. Mas ainda conseguimos adicionar um lance sem utilizar o método `propoe` . Isso acontece porque estamos devolvendo a mesma lista de lances da classe `Leilao` . Ou seja, precisamos devolver outra lista. Para isso, podemos devolver uma cópia da lista de lances na propriedade:

```
@property
def lances(self):
    return self.__lances[:]
```

Bacana! Agora nos testes, basta substituir a adição de um leilão na lista pelo método propõe.

Vamos dar uma olhada na classe `Avaliador` , qual o papel dela? Essa classe trabalha com um leilão e descobre qual o valor do maior e qual o valor do menor lance. Faz sentido este comportamento estar aí? Ou é uma tarefa do `Leilao` conhecer o maior e o menor lance?

A resposta para essa pergunta é depende! As vezes é preciso separar o estado do comportamento. Alguns design patterns fazem isso. Outras vezes não é tão necessário.

No nosso caso, faz sentido o `Leilao` conseguir saber quem é o maior e o menor lance. Portanto, vamos mover o comportamento da classe `Avaliador` para a classe `Leilao` :

```
class Leilao:

    def __init__(self, descricao):
        self.descricao = descricao
        self.__lances = []
        self.maior_lance = sys.float_info.min
        self.menor_lance = sys.float_info.max
```

```
    def propoe(self, lance: Lance):
        if lance.valor > self.maior_lance:
            self.maior_lance = lance.valor
        if lance.valor < self.menor_lance:
            self.menor_lance = lance.valor

        self.__lances.append(lance)

    @property
    def lances(self):
        return self.__lances[:]
```

Após isso, basta removermos o `Avaliador` dos testes e apagarmos esta classe.