05

Refatorando o domínio

Vamos começar a refatorar o código. Antes de tudo, vamos criar uma exceção que representa um lance inválido. Por isso, vamos criar um novo arquivo do Python chamado de `excecoes.py` e nele colocamos o código da classe da nossa exceção:

```
class LanceInvalido(Exception):
    pass
```

Agora basta substituir o lançamento das exceções pela exceção que acabamos de criar:

```
raise LanceInvalido()
```

E temos que alterar o código de testes para que ele espere essa exceção:

```
def test_nao_deve_permitir_propor_lance_com_valor_maior_que_o_da_carteira(vini, leilao):
    with pytest.raises(LanceInvalido):
        # código omitido


def test_nao_deve_permitir_propor_um_lance_em_ordem_decrescente(self):
    with self.assertRaises(LanceInvalido):
        # Código omitido
```

Bacana! Vamos agora começar a isolar as funções. Podemos criar um método `_lance_eh_valido` e dentro desse método, temos as condições que avaliam um lance:

```
class Leilao:

    # código omitido

    def propoe(self, lance: Lance):
        if self._lance_eh_valido(lance):
            if not self._tem_lances():
                self.menor_lance = lance.valor

            self.maior_lance = lance.valor

            self.__lances.append(lance)

    def _lance_eh_valido(self, lance):
        return not self._tem_lances() or (self._usuarios_diferentes(lance) and
                                          self._valor_maior_que_lance_anterior(lance))
```

Esse método tem a função de encapsular os métodos que definem as regras de um lance ser válido ou não. A codificação desses métodos fica próximo a isso:

```
def _tem_lances(self):
```

```
        return self.__lances

    def _usuarios_diferentes(self, lance):
        if self.__lances[-1].usuario != lance.usuario:
           return True
        raise LanceInvalido('O mesmo usuário não pode dar dois lances seguidos')


    def _valor_maior_que_lance_anterior(self, lance):
        if lance.valor > self.__lances[-1].valor:
            return True
        raise LanceInvalido('O valor do lance deve ser maior que o lance anterior')
```

É importante notar também, que refatoramos o código que pegam o valor do maior e do menor lance, como a regra de negócio mudou, o código também teve que ser modificado.

Além da classe leilão, a classe usuário também pode ser refatorada:

```
        def propoe_lance(self, leilao, valor):
            if not self._valor_eh_valido(valor):
                raise LanceInvalido('Não pode propor um lance com o valor maior que o valor da carteira'

            lance = Lance(self, valor)
            leilao.propoe(lance)

            self.__carteira -= valor
```

Criamos um método para isolar a condição que checa se um valor é válido ou não. O código do método fica dessa forma:

```
    def _valor_eh_valido(self, valor):
        return valor <= self.__carteira
```

Conseguimos refatorar bem as classes de domínio. Com essa refatoração, conseguimos deixar o código mais legível, o que ajuda em manutenções futuras no sistema, rodando os testes, todos devem continuar passando.