07

Mãos à obra!

Temos que implementar duas novas regras de negócio. O mesmo usuário não pode dar dois lances seguidos e o próximo lance sempre deve ser maior que o anterior.

Vamos começar por uma abordagem diferente, por isso, começamos pelos testes. Vamos começar implementando um teste para um caso de uso que é quando o leilão não possui lances:

```
def test_deve_permitir_propor_um_lance_caso_o_leilao_nao_tenha_lances(self):
    # código omitido
```

Ao rodarmos esse teste, ele passa. Vamos agora a outro caso de uso. Deve permitir um lance quando o usuário é diferente do anterior:

```
def test_deve_permitir_propor_um_lance_caso_o_ultimo_usuario_seja_diferente(self):
    # implementação omitida
```

Esse teste também já passa com o código que implementamos. Vamos para o terceiro teste. Não deve permitir que o mesmo usuário proponha dois lances seguidos:

```
def test_nao_deve_permitir_propor_lance_caso_o_usuario_seja_o_mesmo(self):
    # implementação omitida
```

Dessa vez o teste falhou! O que temos que fazer? Vamos para a classe de domínio e adicionar a verificação sobre se é o mesmo usuário:

```
def propoe(self, lance: Lance):
    if not self.lances or self.lances[-1].usuario != lance.usuario:
        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)
```

Além de conferir se o último usuário é diferente, precisamos conferir se o leilão não está vazio. Senão é lançado uma exceção.

Bacana! Agora o teste está passando, mas qual o problema dessa abordagem? Não estamos passando um feedback para o usuário. Precisamos que ele saiba que deu um problema na transação, por isso, uma forma é laçar uma exceção:

```
def propoe(self, lance: Lance):
        if not self.__lances or self.__lances[-1].usuario != lance.usuario:
            if lance.valor > self.maior_lance:
                self.maior_lance = lance.valor
            if lance.valor < self.menor_lance:
                self.menor_lance = lance.valor
```

```
                self.menor_lance = lance.valor

            self.__lances.append(lance)
        else:
            raise ValueError('Erro ao propor lance')
```

Legal! Só precisamos alterar o código de testes para testar o lançamento da exceção. Ao herdar da classe `TestCase`, já existe um método que podemos utilizar para testar exceções. Esse método é o `self.assertRaises`:

```
def test_nao_deve_permitir_propor_lance_caso_o_usuario_seja_o_mesmo(self):
    lance_do_gui200 = Lance(self.gui, 200.0)

    with self.assertRaises(ValueError):
        self.leilao.propoe(self.lance_do_gui)
        self.leilao.propoe(lance_do_gui200)
```

Todos os testes voltam a passar! Agora vamos implementar a outra regra de negócio. Um lance deve ser maior que o anterior. Como essa regra é um pouco mais simples, já sabemos como implementá-la, não vamos começar pelos testes, partiremos para a implementação.

Para essa regra funcionar, basta adicionarmos uma condição na classe de domínio:

```
def propoe(self, lance: Lance):
    if not self.__lances or self.__lances[-1].usuario != lance.usuario and lance.valor > self.__lanc
    # implementação omitida
```

Com essa alteração, um dos testes falha! O teste que verifica a inserção em ordem decrescente não passa. Faz sentido a gente ainda ter esse teste no sistema? A regra de negócio mudou, ou seja, não tem mais porque manter esse deste dessa maneira. O que podemos fazer é refatorar esse teste para que ele atenda a nova regra de negócio:

```
def test_nao_deve_permitir_propor_um_lance_em_ordem_decrescente(self):

    with self.assertRaises(ValueError):
        yuri = Usuario('Yuri')
        lance_do_yuri = Lance(yuri, 100.0)

        self.leilao.propoe(self.lance_do_gui)
        self.leilao.propoe(lance_do_yuri)
```

Agora todos os nossos testes estão passando!