08

Mãos à obra!

Antes de começar a escrever os testes, precisamos instalar a biblioteca. Por isso, vamos ao terminal e pedimos para o `pip` instalar a `pytest` para a gente:

```
pip install pytest
```

Com a biblioteca instalada, vamos começar a escrever nossos arquivos de testes. Mas antes, vamos criar um diretório chamado `tests` e mover o `test_leilao.py` para lá. Nesse diretório, criamos outro arquivo chamado `test_usuario.py`. É neste arquivo que vamos colocar o código de testes da classe usuário.

Temos que implementar as novas regras de negócio, mas antes disso, vamos começar pelos testes já para ir conhecendo a biblioteca. Logo, criamos uma função para testar a funcionalidade de subtrair um valor da carteira do usuário:

```
def test_deve_subtrair_valor_da_carteira_do_usuario_quando_este_propor_um_lance():
    # implementação omitida
```

O teste falha! Vamos começar a mexer na classe usuário, antes de tudo, vamos colocar um parâmetro que representará a carteira no método que inicializa os atributos da classe, além disso, vamos criar um método `propoe_lance`, que recebe o leilão e o valor do lance que está sendo proposto:

```
class Usuario:

    def __init__(self, nome, carteira):
        self.__nome = nome
        self.__carteira = carteira

    def propoe_lance(self, leilao, valor: float):
        self.carteira -= valor
        lance = Lance(self, valor)
        leilao.propoe(lance)
```

O teste está passando. Vamos começar a implementar outros testes:

```
def test_deve_permitir_propor_lance_quando_o_valor_eh_menor_que_o_valor_da_carteira():
    # implementação omitida

def test_deve_permitir_propor_lance_quando_o_valor_eh_igual_ao_valor_da_carteira():
    # implementação omitida
```

Bacana! Com o código atual, esses dois testes já passam. Vamos implementar um novo testes, dessa vez, o usuário não pode dar um lance maior do que o valor da carteira. Para esse teste, podemos esperar uma exceção:

```
def test_nao_deve_permitir_propor_lance_com_valor_maior_que_o_da_carteira():
    with pytest.raises(ValueError):
        # implementação omitida
```

Como esse teste não passa, vamos implementar na classe de domínio as regras para fazê-lo passar:

```
def propoe_lance(self, leilao, valor: float):
    if self.carteira >= valor:
            self.carteira -= valor
            lance = Lance(self, valor)
            leilao.propoe(lance)
    else:
            raise ValueError('Nao pode dar lance maior que o valor da carteira')
```

Legal, todos os testes estão passando! Mas, temos alguns códigos repetidos nos métodos de teste. Sabemos que podemos isolar isso de alguma forma. Na biblioteca `unittest` nós utilizamos o método `setUp` herdado da classe `TestCase`. O que fazemos na biblioteca `pytest` já que não herdamos de nenhum lugar?

Na biblioteca `pytest`, basta criarmos uma função e decorar ela dizendo que ela é uma `fixture`;

```
@pytest.fixture
def vini():
    return Usuario('Vini', 100.0)


@pytest.fixture
def leilao():
    return Leilao('Celular')
```

Bacana! Agora, basta passar uma parâmetro no método de testes com o mesmo nome da função:

```
def test_deve_subtrair_valor_da_carteira_do_usuario_quando_este_propor_um_lance(vini, leilao):
    # implementação omitida
```