A dúvida agora é: como podemos testar uma exceção no nosso código? Uma solução possível seria incluir no método testLeilaoVazioNaoPodeSerAvaliado() um try/catch pegando a exceção \DomainException $exception e verificando, com o assertEquals(), que a mensagem sendo exibida é a esperada ("Não é possível avaliar leilão vazio").

public function testLeilaoVazioNaoPodeSerAvaliado()

{

try {

$leilao = new Leilao('Fusca Azul');

$this->leiloeiro->avalia($leilao);

} catch (\DomainException $exception) {

static::assertEquals(

'Não é possível avaliar leilão vazio',

$exception->getMessage());

}

}

Com essa alteração, a execução do teste passará normalmente. Mas o que acontece se deixarmos de lançar uma exceção? Descobriremos isso comentando a linha responsável por isso no método avalia() e executando novamente o teste, que deveria falhar.

if (empty($leilao->getLances())) {

//throw new \DomainException('Não é possível avaliar leilão vazio');

}

Ao rodarmos o teste, entretanto, receberemos uma mensagem diferente informando que os testes foram ok, mas algo está incompleto, já que nenhuma asserção foi executada nesse caso.

OK, but incomplete, skipped, or risky tests!

Tests: 14, Assertions: 27, Risky: 1.

Isso acontece pois o try do método testLeilaoVazioNaoPodeSerAvaliado() foi executado, mas não caiu no catch, portanto nenhuma verificação foi feita. Para esse cenário, podemos incluir no try um método static::fail(), que simplesmente falha, passando a mensagem "Exceção deveria ter sido lançada".

public function testLeilaoVazioNaoPodeSerAvaliado()

{

try {

$leilao = new Leilao('Fusca Azul');

$this->leiloeiro->avalia($leilao);

static::fail('Exceção deveria ter sido lançada');

} catch (\DomainException $exception) {

static::assertEquals(

'Não é possível avaliar leilão vazio',

$exception->getMessage());

}

}

Se executarmos o teste novamente, a mensagem de erro será exibida.

There was 1 failure:

1) Alura\Leilao\Tests\Service\AvaliadorTest::testLeilaoVazioNaoPodeSerAvaliado

Exceção deveria ter sido lançada

C:\Users\Rodrigo\Desktop\php-testes\leilao\tests\Service\AvaliadorTest.php:76

FAILURES!

Tests: 14, Assertions: 28, Failures: 1.

Essa lógica funciona, e não está necessariamente errada, mas não é a solução mais elegante, pois o próprio PHPUnit nos fornece funcionalidades para realização desse tipo de teste. Portanto, vamos apagar todo o try/catch que criamos, algo que poderia facilmente ser chamado de "gambiarra", e partiremos para outra solução.

O PHPUnit nos fornece um método chamado excepctedExcetion() com o qual podemos informar que determinado código precisará lançar uma exceção e que tipo de exceção é essa - no caso, uma \DomainException.

public function testLeilaoVazioNaoPodeSerAvaliado()

{

$this->expectException(\DomainException::class);

$leilao = new Leilao('Fusca Azul');

$this->leiloeiro->avalia($leilao);

}

Se executarmos o teste dessa forma, ele não passará e teremos uma mensagem *"Failed asserting that exception of type "DomainException" is thrown"*, já que ainda não removemos o comentário no lançamento da exceção em avalia(). Vamos corrigir isso.

if (empty($leilao->getLances())) {

throw new \DomainException('Não é possível avaliar leilão vazio');

}

Com uma nova execução, nosso teste passará corretamente. Também podemos verificar se a mensagem está correta usando o método $this->exceptedExceptionMessage():

public function testLeilaoVazioNaoPodeSerAvaliado()

{

$this->expectException(\DomainException::class);

$this->expectExceptionMessage('Não é possível avaliar leilão vazio');

$leilao = new Leilao('Fusca Azul');

$this->leiloeiro->avalia($leilao);

}

Nossos testes continuarão passando, agora com uma verificação a mais. Assim, temos uma garantia bem mais simples e elegante de que nosso código lançará uma exceção. Entretanto, estamos quebrando o nosso modelo *arrange, act, assert*, já que estamos fazendo o *assert* antes do *arrange* e do *act*.

Isso pode ser um problema para alguns, principalmente aqueles que necessitam que padrões sejam seguidos à risca. Nesses casos, é possível que o indivíduo prefira implementar com try/catch. Deve-se pensar que não existe nenhuma desvantagem em construir o código dessa forma, já que estamos verificando o lançamento de uma exceção e qual mensagem está sendo exibida, ao mesmo tempo em que ganhamos em legibilidade.

Com isso, conseguimos implementar a regra de que um leilão não pode ser avaliado se não tiver lances, e ela está sendo testada ao mesmo tempo em que todo o resto do nosso sistema funciona. Inclusive, se por algum motivo tivéssemos escrito essa funcionalidade de forma errada, diversos dos nossos testes anteriores falhariam, pois teríamos quebrado o nosso código. Isso é chamado de **teste de regressão**.

Ainda temos outra questão a refletir sobre a nossa regra de negócios. Será que realmente devemos ignorar os lances quando o usuário tentar dar lances consecutivos ou quando um usuário der mais que 5 lances? Não faria mais sentido lançarmos exceções também nesses casos? É isso que faremos no próximo vídeo.