Aproveitaremos esse momento para recapitular as mudanças que fizemos no último vídeo, de modo a avançar com mais clareza no nosso desenvolvimento. Tínhamos vários testes que, em sua essência, se repetiam, como o testAvaliadorDeveEncontrarOMaiorValorDeLancesEmOrdemCrescente() e o testAvaliadorDeveEncontrarOMaiorValorDeLancesEmOrdemDecrescente(). Ou seja, tínhamos dados diferentes, mas a estrutura dos testes era a mesma.

Pensando nisso, separamos os dados em métodos específicos e os passamos para os testes por meio dos *data providers*. Aprendemos, então, que é possível utilizar apenas um *data provider* que fornece todos os dados, onde cada linha do array retornado pelo método que carrega os dados gera um teste, ou vários *data providers* individuais, que foi a maneira que preferimos manter.

Também aprendemos que o formato de retorno dos dados de um *data provider* não é muito usual, com cada índice do array gerando um teste e cada item desse array será um parâmetro a ser passado para o teste. Assim, conseguimos diminuir o número de métodos de teste, além de simplificar o nosso código. Também poderemos criar outros dados de entrada, caso necessário, sem alterar o código de teste.

Nossos métodos de teste agora possuem uma seção *arrange* que consiste apenas na criação de um $leiloeiro e da avaliação do leilão. Depois, passamos para as seções *act* e *assert*, nas quais testamos efetivamente o nosso código. Mas repare que também podemos extrair o código que cria um $leiloeiro.

Imagine que nossa aplicação cresça bastante e o avaliador precise de um novo objeto, como FormaDeAvaliacao. Se estivermos criando esse $leiloeiro em cada um dos métodos, perderemos em agilidade, pois teremos que editar cada um dos métodos que faz essa criação.

Sendo assim, no início da classe AvaliadorTest, criaremos um atributo privado $leiloeiro e um método criaAvaliador() que, por enquanto, apenas atribui a ele um novo Avaliador.

class AvaliadorTest extends TestCase

{

private $leiloeiro;

public function criaAvaliador()

{

$this->leiloeiro = new Avaliador();

}

//...

Nos testes, passaremos a chamar o método $this->criaAvaliador(), e o nosso $leiloeiro passará a ser buscado da instância dessa classe de testes (com $this->leiloeiro). Começaremos alterando o testAvaliadorDeveEncontrarOMaiorValorDeLances().

/\*\*

\* @dataProvider leilaoEmOrdemAleatoria

\* @dataProvider leilaoEmOrdemCrescente

\* @dataProvider leilaoEmOrdemDecrescente

\*/

public function testAvaliadorDeveEncontrarOMaiorValorDeLances(Leilao $leilao)

{

// Arrange - Given / Preparamos o cenário do teste

$this->criaAvaliador();

// Act - When / Executamos o código a ser testado

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

$maiorValor = $this->leiloeiro->getMaiorValor();

// Assert - Then / Verificamos se a saída é a esperada

self::assertEquals(2500, $maiorValor);

}

E repetiremos o processo nos dois outros testes.

/\*\*

\* @dataProvider leilaoEmOrdemAleatoria

\* @dataProvider leilaoEmOrdemCrescente

\* @dataProvider leilaoEmOrdemDecrescente

\*/

public function testAvaliadorDeveEncontrarOMenorValorDeLances(Leilao $leilao)

{

// Arrange - Given / Preparamos o cenário do teste

$this->criaAvaliador();

// Act - When / Executamos o código a ser testado

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

$menorValor = $this->leiloeiro->getMenorValor();

// Assert - Then / Verificamos se a saída é a esperada

self::assertEquals(1700, $menorValor);

}

/\*\*

\* @dataProvider leilaoEmOrdemAleatoria

\* @dataProvider leilaoEmOrdemCrescente

\* @dataProvider leilaoEmOrdemDecrescente

\*/

public function testAvaliadorDeveBuscar3MaioresValores(Leilao $leilao)

{

$this->criaAvaliador();

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

$maiores = $this->leiloeiro->getMaioresLances();

static::assertCount(3, $maiores);

static::assertEquals(2500, $maiores[0]->getValor());

static::assertEquals(2000, $maiores[1]->getValor());

static::assertEquals(1700, $maiores[2]->getValor());

}

Após isso, executaremos vendor\bin\phpunit --color tests no terminal para confirmarmos que os testes continuam passando. Agora conseguimos separar a criação do nosso avaliador, permitindo que alterações sejam feitas em um único ponto. Porém, note que, nos testes, apenas trocamos uma linha por outra. O ideal seria não precisarmos chamar o criaAvaliador(), mas sim sempre executarmos a sua criação antes de cada um dos testes.

O PHPUnit nos fornece essa funcionalidade por meio do método protegido setUp(), que precisa retornar void. Todo código que fizer parte desse método sempre será executado antes que os testes sejam rodados.

class AvaliadorTest extends TestCase

{

private $leiloeiro;

protected function setUp(): void

{

$this->leiloeiro = new Avaliador();

}

//...

Agora que temos um método que sempre cria um leiloeiro, podemos simplesmente remover essa criação dos nossos testes.

/\*\*

\* @dataProvider leilaoEmOrdemAleatoria

\* @dataProvider leilaoEmOrdemCrescente

\* @dataProvider leilaoEmOrdemDecrescente

\*/

public function testAvaliadorDeveEncontrarOMaiorValorDeLances(Leilao $leilao)

{

// Act - When / Executamos o código a ser testado

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

$maiorValor = $this->leiloeiro->getMaiorValor();

// Assert - Then / Verificamos se a saída é a esperada

self::assertEquals(2500, $maiorValor);

}

Com isso, nos testes que construímos até o momento, deixamos de ser responsáveis por criar o cenário, passando a prepará-lo no setUp() e a receber os dados por meio dos *data providers*. Executando os testes no terminal, todos passarão corretamente.

Devemos nos lembrar que códigos de teste também precisam ser sucintos, organizados e legíveis. Pensando nisso, refatoramos o nosso AvaliadorTest de modo a eliminar as redundâncias testar o essencial de maneira mais efetiva. Além disso, a separação entre a preparação do ambiente, os dados e os testes em si nos traz diversas vantagens.

Porém, note que estamos criando os usuários em cada um dos *data providers*. Será possível fazermos isso no setUp(), fazendo que os *data providers* só usem tais usuários? Falaremos sobre isso no próximo vídeo.