Continuaremos nosso desenvolvimento com os testes já passados para o xUnit.

Por enquanto, os lances estão de forma desordenada. Criaremos uma nova verificação para o sistema de leilão que recebe ofertas ordenadas por valor sob o método LeilaoComLancesOrdenadosPorValor() no topo de LeilaoTestes, sem esquecer de adicionar a anotação [Fact] da mesma maneira que já vimos.

Na declaração, copiamos e colamos o mesmo código do corpo do teste com vários lances. Nas linhas de leilao.RecebeLance(), ordene-as por valor dado. Repare que o Gerenciador de Testes já reconheceu o novo método.

Criamos uma verificação antes da anterior chamada LeilaoComTresClientes(), recebendo três ofertas diferentes. Copie e cole novamente o código no corpo do novo método, inserindo mais uma variável beltrano para representar o terceiro cliente após fulano e maria.

Em seguida, adicione mais um lance feito por Beltrano no valor de 1400 e depois altere para o mesmo valor em valorEsperado. Ainda, faremos um teste para identificar o nome da pessoa com a maior oferta, neste caso, beltrano através de uma nova linha Assert.Equal().

public class LeilaoTestes

{

[Fact]

`LeilaoComTresClientes()`

var leilao = new Leilao ("Van Gogh");

var fulano = new Interessada("Fulano, leilao);

var maria = new Interessada ("Maria", leilao);

var beltrano = new Interessada ("Beltrano", leilao);

leilao.RecebeLance(fulano, 800);

leilao.RecebeLance(maria, 900);

leilao.RecebeLance(fulano, 1000);

leilao.RecebeLance(maria, 990);

leilao.RecebeLance(beltrano, 1400);

leilao.TerminaPregao();

var valorEsperado = 1400;

var valorObtido = leilao.Ganhador.Valor;

Assert.Equal(valorEsperado, valorObtido);

Assert.Equal(beltrano, leilao.Ganhador.Cliente);

}

Podemos ter mais de um Assert.Equal() na verificação relacionada ao ganhador, pois este método possui sobrecarga para vários tipos de dados diferentes como double e a referência Interessada nestes casos.

Executamos os testes para ver que todos estão aprovados no Gerenciador.

Existe ainda algumas questões; quantos testes seriam necessários para assegurar o funcionamento de um programa? Veremos adiante. Mas por enquanto, nosso código tem muitas duplicações, deixando-o extenso e repetitivo.

No próximo passo, veremos como deixá-lo mais limpo e organizado.

Neste passo, vamos responder à pergunta anterior descobrindo quando terminar os testes, ou seja, confirmar se a funcionalidade é suficientemente confiável e pronta para produção.

Para isso, temos duas dicas: a primeira é procurar orientação de profissionais mais experientes e a segunda é usar o padrão de teste sob três sessões como já vimos. Além do AAA, temos o Given, When e Then.

Começando por LeilaoComTresClientes(), aplicamos o conceito do segundo padrão descrevendo as etapas de verificação em comentários no próprio código com //, para esclarecer mais cada situação do programa e o que esperamos como resultado do teste em cada um, podendo ser transcrito para qualquer forma de registro de fluxo de trabalho que preferir para não poluir nosso código, como o Trello por exemplo.

Faça a mesma descrição de cada cenário diferente para os quatro testes que implementamos. Analisando-os, há situações quase idênticas acontecendo, como as expectativas de maior valor esperado.

Então podemos reduzir os quatro cenários em apenas um; nesta análise, observando os valores de entrada e saída esperada, trabalhamos com uma técnica de teste chamada classes de equivalência ou equivalence partitioning ou ainda equivalence class partitioning (ECP), a qual consiste em dividir os dados em várias partições para tornar os testes mais eficientes.

Da forma como está, os quatro testes lidam com uma mesma partição, ou seja, está bem ineficiente. Não importa o cenário, a expectativa final sempre é a de maior valor com a exibição do cliente ganhador dos lances e por isso o LeilaoComTresClientes() é o mais completo dentre todos. Neste, precisamos que haja pelo menos um lance com qualquer número de clientes.

Esta análise só pode ser feita com a experiência de fazê-la, contando idealmente com uma mentoria, sempre focando nos valores. Vale a pena pesquisar uma outra técnica chamada análise dos valores de fronteira ou boundary-value analysis.

De volta às anotações, é preciso pensar em um outro cenário possível onde nenhum lance é dado.

Resolveremos sobre o cenário onde nenhum lance é dado no leilão.

Neste caso, quem seria o ganhador? É uma decisão a ser tomada pelo responsável pelo produto final de forma arbitrária. Neste caso, imaginemos que a oferta vencedora tem valor 0 para podermos fazer um teste baseado neste cenário.

Então, no Visual Studio, adicionamos mais um método público de teste dentro de LeilaoTestes com anotação [Fact] chamado LeilaoSemLances(). Dentre deste, copiamos e colamos o corpo de código das outras verificações para podermos realizar as devidas alterações.

Começamos retirando as linhas dos clientes interessados e das ofertas. Depois, altere o valor de valorEsperado para 0.

public void LeilaoSemLances()

{

var leilao = new leilao ("Van Gogh");

leilao.terminaPregao();

var valorEsperado = 0;

var valorObtido = leilao.Ganhador.Valor;

Assert.Equal(valorEsperado, valorObtido);

}

Rodamos os testes no Gerenciador para observarmos o resultado. Vemos que esta última verificação apresenta falha, e devemos ver suas especificações exibidas.

Há uma exceção nativa do .Net detectada chamada System.InvalidOperationException, apontando que a sequência não contém elementos, ou seja, o enumerado de lances não possui nenhum item dentro.

Pela pilha de execução vemos que isso foi feito justamente quando chamamos o método Last() dentro de TerminaPregao(). Na documentação deste método problemático, o programa indica que há exceções que podem ser lançadas justamente por não haver elementos nesta sequência.

Para resolver, alteramos o uso de Last() para LastOrDefault(), o qual se diferencia do primeiro por verificar se está vazio, e se estiver, retorna um objeto default qualquer. Para um enumerado de lances, definimos o retorno com o método DefaultIfEmpty() criando um objeto a ser usado neste caso com cliente null e valor 0.

public void TerminaPregao()

{

Ganhador = Lances

.DefaultIfEmpty(new Lance(null, 0))

.OrderBy(l => l.Valor)

.LastOrDefault();

}

Com essas alterações, executamos novamente no console para analisar os resultados. Dessa vez, tudo funciona como esperado.

Vale lembrar que é importante conhecer, explorar e estudar a API do .NET, pois são nesses momentos que este tipo de conhecimento é necessário para resolver questões apresentadas no código de maneira eficiente.

Nossa funcionalidade de avaliação do leilão possui duas classes de equivalências, sendo que ainda temos cinco testes implementados. A seguir, veremos como reduzir e limpar nosso código.