Testes Parametrizáveis:

Podemos observar que os métodos de calcular desconto de filmes da locação seguem estruturas iguais, alterando somente a massa de entrada de dados(Quantidades de filmes na lista) e a massa de dados de saida que deve ser validada na etapa de verificação(Resultado dos Asserts).

Então vamos usar uma técnica diferente para testar o funcionamento do método. Criamos uma nova classe de teste apenas para o calculo da locação, extraimos os dados de entrada que são as listas em uma única variável de classe, o mesmo com o valor da assertiva que agora é uma única variável do tipo Double e lógicamente uma variável da classe de LocacaoService.

Nosso @Before método de setUp( ) vai instânciar a variável de serviço da LocacaoService e nosso método de teste agora terá o Usuario como parte do cenário(Given), o service como parte da execução(When) e a assertiva usando nosso Double valorLocacao(Then).

Data Driven Test:

Com esse único teste vamos conseguir o resultado de todo aquele conjunto de testes. Temos toda a estrutura pronta, precisamos apenas inserir os dados que irão guiar esses testes. Como essa classe não será executada da maneira normal do JUnit, usamos uma anotação **@RunWith(Parameterized.class**) para que ele saiba como deve executar os testes da classe, ou seja, de maneira parametrizáda.

A brincadeira nesse ponto já começa a ficar sinistra, fizemos um método para ser o parâmetro do teste, nele falamos que o retorno é uma **Collection<Object[]>** e passamos nossos filmes e os valores que esperamos em uma matriz de objeto para serem usados como parametros do teste. Para que o JUnit entenda que o método é o parâmetro, usamos a anotation **@Parameters** e o método precisa ser **static** assim como as variáveis dele.

Agora é necessário fazer um link com os parâmetros definidos no método pegarParametros( ) com as variáveis que estão sendo utilizadas no teste. Para isso usamos a anotação **@Parameter** , a lista de filmes é o primeiro parâmetro e o valor é o segundo. Então a lista de filmes recebe a annotation @Parameter e o valorLocacao recebe a annotation @Parameter(value = 1) isso porque nos arrays de pegarParametro( ) a posição 0 do arrray é a lista de filmes e a posição 1 são os valores.

Adicionamos um terceiro parâmetro para legibilidade do teste.

Com essas configurações podemos deixar nossa classe de testes de serviço bem mais enxuta, e agora todos os métodos que seguem a mesma estrutura de desconto por quantidade de filme ficaram mais "dinâmicos". Percebemos que a complexidade do Data Driven Test é alta, mas agora temos uma estrutura que pode ser escalavel a medida que forem especifícadas mais demandas de desconto, ou caso elas mudem.