DataBuilders:

Criar objetos para cenários de testes muitas vezes é uma tarefa repetitiva e chata, para resolver isso existe um padrão que **centraliza a criação das entidades** e traz mais legibilidade aos testes, esse padrão é o **Test DataBuilder.**

No package de builder, criamos o **UsuarioBuilder** :

Seu **contstrutor é privado** para que ninguém possa criar instâncias do UsuarioBuilder fora da própria classe**(só o builder pode instânciar ele mesmo)**.

O método **"umUsuario()"** que retorna um **UsuarioBuilder** é static para que ele possa ser usado externamente sem uma instância da classe de UsuarioBuilder, esse método é a **porta de entrada para a criação de um usuário.** É por isso que nele instânciamos um novo builder**(não tem outra forma de se criar que não seja por esse método),** inicializamos a construção do atributo private de usuario da classe e já estamos **setando o usuario do builder com um atributo qualquer**(nesse caso o nome). Como no final retornamos a **instância do builder**, podemos chamar outros **métodos do próprio builder**, isso se chama **Chaining method**(método encadeado).

O método **"agora()"** é o método que usaremos no encadeamento, ele apenas retorna o usuario que foi montado por esse builder.

Agora todo teste que use uma instância de usuário poderá usar o BuilderUsuario, e seguindo o método encadeado nós temos então : **"UsuarioBuilder.umUsuario( ).agora( )"** , um ponto a se notar é a legibilidade.

FilmeBuilder:

Criamos um **FilmeBuilder** nos mesmos moldes do **UsuarioBuilder**(afinal de contas é um padrão), então temos sua **variável de classe do tipo Filme**, seu **construtor privado** para que ele não seja criado fora da classe, seu método **public static que retorna o builder** valores "padrões" pré setados e o seu método "agora()" que retorna um **Filme para ser encadeado com o método "umFilme()".**

Da mesma forma podemos usar agora o builder nos testes que precisam de um Filme, então temos : **"FilmeBuilder.umFilme( ).agora( ).**

Builders que precisam de atributos com valores diferentes nos testes:

Fazendo a **refatoração dos testes usando o builder** recém criado do filme vamos perceber que **alguns testes quebram**, isso **porque existe um teste que lança uma exceção caso o filme esteja sem estoque** porém setamos no builder **por padrão que todo filme no atributo estoque tem valor 2**.

Como resolvemos isso sem quebrar os outros testes que utlizam o builder padrão? bem simples, **devemos criar um novo método que altere somente o atributo necessário para execução do teste específico**.

Então criamos um método **"semEstoque( )"**do tipo FilmeBuilder, que seta o estoque da variável de instância filme para 0, como esse método **é um método de instância**, **retornamos a própria instância com o this**.

Nosso teste de lançar uam exceção de filme sem estoque agora usa um: **"FilmeBuilder.umFilme( ).semEstoque( ).agora( )"**, mais uma vez um ponto a ser ressaltado é a legibilidade do código, invés da gente precisar analisar o construtor do builder, podemos ler os métodos para entender o que ele quer passar.

O que permite esse encadeamento:

É o padrão **"chaining method"** que permite o encadeamento, ao final da criação do objeto **depois de encadear todos os métodos de criação** informamos ao builder que **finalizamos chamando o método "agora()"**, que retorna a variável de instância.

Então usamos o **construtor para iniciar nossa criação do objeto,** encadeamos os métodos criados para **ir definindo como vai ser a construção do objeto**, então podemos usar os valores padrões do construtor, ou ir adicionando valores neccessários através dos metodos, independete da maneira que escolher, ao final chamamos o método que instância a variável da classe de fato.