04

Injeção de dependências

Transcrição

[00:00] O que precisamos fazer agora é entender um detalhe que é muito importante para a continuação desse treinamento e que ainda vamos utilizar bastante.

[00:08] Na classe “HomeController”, a primeira coisa é que não criamos instâncias dessa classe. Ela é gerenciada pelo próprio Spring e, sendo assim, esse “EndityManager” tem um atributo necessário para essa classe - ele também é gerenciado pelo Spring e injetado dentro dessa classe.

[00:28] Isso quer dizer o seguinte: se você tentar criar uma instância de “HomeController”, para ele funcionar, você vai ter que fazer na mão mesmo como estou fazendo agora, criar todas as dependências que ele precisa para executar.

[00:44] Por exemplo: “HomeController.home”, que é aquele método de criamos que recebe um “model”. Esse “model” é o que nós também temos que passar para ele. Eu vou criar uma instância de “model”. Deixe-me só ver uma instância, “ExtendedModelMap”. Estou só criando. É um mapa de valores.

[01:12] E logo em seguida, estou tentando fazer na mão esse aqui para ver se eu consigo. Na verdade, vou tentar imprimir um “model”. O que acontece? Quando chamamos um método “home” passando o “model”, esse método aqui dentro adiciona como atributo do “model” uma lista de pedidos. Eu só quero tentar acessar essa lista de pedidos. Beleza! Como eu estou no método “main”, para eu executar, é só clicar com o botão direito “Run As”, “Java Application” ver o que acontece no console.

[01:48] Tomamos um “NullPointer” na linha 25. Por quê? Criamos uma instância “homeController” e chamamos o método “home”. Na hora que chamamos o método “home”, ele tenta usar o “EndityManager” para criar uma “query”, acessar o banco e tal.

[02:02] Esse “EndityManager” não foi instanciado. Ninguém deu “new” nesse “EndityManager”. Então não adianta criarmos só a classe “homeController” e sabermos passar os parâmetros para o método que estamos chamando, nós também temos que resolver os atributos dele.

[02:17] Então imagine fazer isso na mão! O que o Spring está fazendo aqui? Ele não só está criando as instâncias de “homeController”, mas está vendo o seguinte: essa é uma instância de “homeController”. Uma instância de “homeController” depende do “EndityManager”, que é alguém que vai no banco de dados.

[02:33] E para ir no banco de dados ela precisa de uma “collection”. Então o Spring tenta resolver uma “collection” também. Só que na hora de criar uma “collection” precisa de propriedades, que é esse “application.properties”.

[02:46] Tudo isso o Spring está resolvendo. Não é que o Spring saiba como criar essas instâncias, mas ele sabe ir até alguém que sabe criar essas instâncias e ele injeta as dependências quando é necessário. Por exemplo: para podermos utilizar mais de injeção de dependência de forma que utilizamos isso para organizar o nosso código. O “homeController” está com muita responsabilidade. Ele sabe ir no banco de dados e fazer consulta. Ele tem dependências do tipo “EndityManager”.

[03:22] Se abrirmos as dependências, tem dependências “EndityManager”, “PersistenceContext” e “Query”. Tem várias dependências porque essa classe está com muita responsabilidade. Então o que podemos fazer é criarmos um repositório onde ele vai ficar com essa lógica de fazer consultas ao banco de dados, que chamamos de “Repository”.

[03:45] É uma classe com a responsabilidade de acessar um repositório de informações de dados. Então eu vou criar um “PedidoRepository” como uma classe mesmo. Essa classe vai retornar uma lista do tipo “Java.util List” do tipo “Pedido” e vou criar uma “recuperaPedidos”. Na verdade, “TodosOsPedidos”.

[04:15] A implementação que fizemos recupera todos os pedidos. E aí, o que eu vou fazer? Eu vou pegar esse “PersistenceContext” e jogar lá dentro de “PedidoRepository”. Porque o “PedidoRepository” vai utilizar esse aqui para fazer a consulta. Em vez de ficar com essa instância aqui dentro, eu vou só retornar diretamente.

[04:37] Pronto! Então esse “PedidoRepository” é quem está utilizando o “EndityManager” - de forma que “homeController” não tem mais esse acesso a “EndityManager”. Ele tem acesso ao “PedidoRepository”. Então eu vou injetar “PedidoRepository” e chamar só de “Repository” e vou tentar utilizar isso para recuperar todos os pedidos. Será que vai funcionar?

[05:02] Vou dar um “[LOW CONFERIBLE]”, chamar de “Pedidos” e pronto. Ou seja, essas dependências “Javax.Persistence” não tem mais e isso aqui também não. Pronto, ele tem menos dependências agora! O “Repository” acabamos colocando dentro do próprio “controller”. Eu vou alterar isso. Vou fazer um “Refector” e dar um “Move” e criar um novo pacote chamado de “Repository”.

[05:39] Estamos organizando as coisas de uma forma diferente, mas nesse projeto vamos fazer criando esses pacotes mesmo. Então eu criei o “PedidoRepository”. O “PedidoRepository” é agora importado aqui, é mais uma dependência. Então a “homeController” depende de “Pedido”, depende de “PedidoRepository” e tem essas outras dependências de anotação, “Controller” e tudo o mais que tem haver com ela mesmo. Beleza, isso está ok!

[06:06] Vamos ver como ele vai executar. Vou subir a classe de “MudiApplication” e ver se ele consegue funcionar corretamente, se ele consegue acessar o repositório. Vamos lá! Nós tomamos um “NullPointer” nc “homeController” na linha 19. Vamos olhar na linha 19 de “homeController”.

a linha onde usamos o “Repository” para chamarmos o método e recuperarmos todos os pedidos. Então, basicamente o que está acontecendo é que esse repositório está nulo. Beleza!

[06:41] Ele está nulo porque o Spring não vai automaticamente injetar para você uma dependência só porque você colocou um atributo ali. Para o Spring injetar para você, você vai colocar o “AutoWired”, que é uma anotação nova.

[06:55] Não vimos ela antes. Ela serve para você pedir para o Spring – “eu quero uma instância de ‘PedidoRepository’! ” Beleza, vamos ver o que acontece! Vou derrubar e vou tentar subir a aplicação de novo. Tomei um erro aqui. Vamos lá!

[07:19] “APPLICATION FAILED TO START”, “Description” da falha é que no “field repository” na classe “homeController” requer uma instância do tipo “PedidoRepository” que o Spring não conseguiu resolver. Ou seja, nós estamos pedindo para o Spring criar uma instância de “PedidoRepository” e colocar aqui. O Spring está dizend:, “eu não sei fazer isso”. Olhe só, a classe “homeController” é instanciada pelo Spring porque ela foi anotada com esse “@Controller”.

[07:54] Então o que podemos fazer é pegar esse “Repository”, essa classe, e pedirmos para o Spring gerenciar ela também. É só colocarmos “@Controller” ou utilizarmos uma anotação que tem uma semântica mais interessante, que é a “Repository”.

[08:12] O Spring também tem essa anotação. Uma anotação que representa o seguinte: você está dizendo para o Spring – “essa classe aqui é um repositório e eu quero que você gerencie essa classe e crie instâncias toda vez que alguém pedir”. O Spring não vai fazer isso automaticamente.

[08:33] Então eu vou subir de novo a aplicação e ver se ele consegue resolver. Beleza, não deu erro para iniciar! Vou apertar a tecla “F5” e agora ele vai conseguir. Ou seja, agora o Spring está gerenciando a classe “PedidoRepository” e criando instâncias dessa classe.

[08:53] Quando o Spring for criar uma instância de “PedidoRepository”, ele vai ver que essa classe tem um atributo do tipo “EndityManager”, na qual ele também sabe resolver. Então ele não vai ter nenhum problema para criar instâncias do tipo “Repository”, assim como eu já mostrei.

[09:10] E aí fica mais interessante porque em termos de organização da nossa aplicação, nós não temos toda aquela lógica de irmos no banco de dados aqui dentro do nosso “Controller”. As dependências também mais simplificadas. Todas essas dependências são específicas de “Controllers” mesmo e as outras dependências que ele tem, tem haver com o negócio da nossa aplicação. Eu acho que está bom, por enquanto.

[09:37] A “PedidoRepository” tem dependências que tem haver com repositório mesmo, com o fato do que ela é. Por exemplo: ela precisa do um “EndityManager” que ela vai usar para acessar o repositório e tem dependência de uma classe de negócio, a classe “Pedido”. Eu acho que em termos de dependências está bom também. Ou seja, a nossa aplicação está bem organizada - e aí é interessante, veja que não estamos precisando criar instâncias de “PedidoRepository”.

[10:09] Essas instâncias estão sendo criadas automaticamente pelo Spring. Então isso é injeção de independência. Nós vamos ver mais sobre isso e vamos usar posteriormente bastante injeções de dependências, tanto para organizarmos o nosso código quanto para simplificarmos.

[10:25] Então, até o próximo vídeo!