05

# Adicionando Maven a um projeto existente

## Transcrição

[00:00] Agora já aprendemos como criar um projeto utilizando o Maven no Eclipse do zero. Só que pode acontecer também de você já ter uma aplicação existente que não tem o Maven e você querer adicionar o Maven nesse projeto existente. No vídeo de hoje vamos aprender como fazemos isso.

[00:19] Eu estou com meu Eclipse aqui, tem aquele projeto “loja” com aplicação Maven que criamos do zero e tem esse outro projeto aqui, chamado “controle-produtos” - que é uma aplicação que não utiliza Maven. É uma aplicação Java web tradicional.

[00:32] Como fazemos para adicionar o Maven nesse projeto? Eu vou ter que adicionar o ‘pom.xml’ na mão. O que eu preciso fazer? No caso do Eclipse bastaria você clicar com o botão direito do mouse no projeto: “controle-produtos > Configure > Converte to Maven Project”.

[00:50] Se você clicar nessa opção, “Convert to Maven Project”, ele vai converter esse projeto para uma aplicação Maven, vai criar o “pom.xml”, vai dar uma analisada na estrutura desse projeto e tentar extrair algumas coisas automaticamente.

[01:02] Vamos ver o que vai acontecer. Em “Group id” preciso dizer qual é o “group id”. Isso ele não vai conseguir descobrir sozinho. Vamos colocar “br.com.alura”. O “Artifact id” coloca o mesmo nome do projeto: “controle-produtos”. Em “Packaging” é WAR . Ele já detectou que é uma aplicação web. Clicamos em “Finish” e ele transforma em uma aplicação Maven, cria o “pom.xml” e vem com esse “pom.xml” com bastante coisas declaradas.

[01:29] Só que ele não troca aquela estrutura de diretórios porque ele não sabe exatamente qual vai ser a estrutura de *source folders* do nosso projeto e como que seria a migração - ele não faz isso automaticamente. Ele tenta descobrir analisando o código-fonte da aplicação.

[01:46] Se olharmos aqui o "pom.xml" colocou aqui uma *tag* `build` e aqui ele falou: "encontrei um diretório 'src' e ele já marca que o 'src' é *source folder* padrão".

[01:59] Ele não faz aquela descoberta, aquele "src/main/resource" não extrai isso automaticamente. O diretório de teste ele descobriu que é o diretório 'test'. É aquela configuração, '<sourceDirectory src' e 'test', que eu tinha comentado no último vídeo. Se você não quiser seguir a estrutura de diretórios padrão do Maven, daria para você personalizar. Ele faz isso quando você cria um projeto do zero.

[02:21] Também configurou um *plugin* para aplicação web e configurou o diretório 'WebContent', 'WebContent', como sendo o diretório web da aplicação. Aqui ele não está seguindo a estrutura de um diretório Maven padrão.

[02:33] Já é uma aplicação Maven, porém ela não está naquela estrutura de diretórios tradicional do Maven. Você poderia deixar assim, que é o jeito mais fácil. Já está com o Maven, é só seguir daqui para frente. Ou o recomendado, que é um pouco mais "chato"; você teria que adaptar esse projeto para seguir aquela estrutura de diretórios padrão do Maven.

[02:53] Vamos ver como que poderíamos seguir a estrutura padrão do Maven, só o projeto para ficar certo e bonito. Vou clicar com botão direito do mouse no projeto, "controle-produtos > Build Path > Configure Build Path" e aqui tem os dois: *source folders*, "src" e "test".

[03:10] Eu vou remover os dois clicando em "Remove > Apply", e o projeto vai ser atualizado, aqueles dois *source folders* vão sumir. Precisamos adicionar

manualmente aqui os *source folders*. Essa é uma parte meio chata, mas se quisermos seguir a estrutura padrão, que é o recomendado, temos que seguir.

[03:30] Clique em “Create New Folder > src/main/java> Finish > OK", já criou o primeiro.

[03:42] “Add Folder > Create Nem Folder > src/main/resources > Finish > OK", criou o segundo. Agora, os de teste, "Add Folder > Create New Folder > src/test/java > Finish > OK". E o último, "Add Folder > Creat New Folder > src/test/resources > Finish > OK”.

[04:12] Estão aí os quatro *source folders*, porém vamos ter que fazer um ajuste aqui no *source folder* de teste. Para dizermos certo aqui no Eclipse para o Maven que esse *source folder* é o *source folder* de teste, vamos expandir o “controle-produtos/src/test/java (new)”.

[04:28] E perceba que tem essa última opção: “Contains test sources: No”. Se você der dois cliques, ele troca para “Yes”. Como aqui vão está os códigos de teste, precisamos marcar essa opção como “Yes”.

[04:41] Automaticamente precisamos trocar o “Output folder: (Default output folder)” para dizer onde que ele vai jogar as classes compiladas de teste. Damos dois cliques no “Output folder: (Default output folder) > Specific output folder (path relative to 'controle-produtos')" e escrevemos no campo de texto: "target/test-classes”, geralmente o diretório de código compilado de teste é esse daqui, “target/test-classe”. Clicamos em “OK” e ele já adapta.

[05:16] Aqui no “src/test/resources/” a mesma coisa. Dois cliques em “Contains test sources: Yes > Output folder > Specific output folder (path relative to 'controle-produtos') > target/test-classes > OK > Apply”.

[05:33] Nos diretórios de *resources*, no campo “Exclused (None)” precisamos trocar tanto no “src/main/resources” quanto no “src/test/resources”. Vou dar dois cliques com o botão esquerdo do mouse no “Exclused > Exclusion

patterns: > Add", inserir “ ** ” e clicar em "OK > Finish”. A mesma coisa no “src/test/resources”, “Exclused > Exclusion patterns: > Add", inserir “ ** ” e clicar em "OK > Finish > Apply > Apply and Close” e está finalizado.

[06:08] Agora, o nosso projeto está seguindo certo a estrutura de diretórios do Maven com os *source folders* todos configurados corretamente. Porém ele perdeu as nossas classes. Na verdade, ele não perdeu, aqui embaixo eu tenho “src”, dentro dele tem essa pasta “br”. Eu posso clicar em “src/main/java”, selecionar e arrastar para o “src/main/java”. Pronto, já jogou para cá o nosso pacote, tudo certo!

[06:34] E, dentro do "src" esses “hibernate.properties", "log4j.xml", "messages.properties” seriam arquivos de configuração. Vou selecioná-los e arrastá-los para o “src/main/resources”. Pronto. Tem essa pasta “test”, dentro dela tem o “br”, vou arrastar o “br” e jogar para “src/test/java”.

[06:55] Posso apagar essa pasta “test”, posso apagar essa pasta “build”, que era onde ele compilava o projeto anteriormente, vou deletar ela. Ficou o “src/main”, o “src/test” e aqui o “target”. O “WebContent”, na verdade, vai ficar dentro de “src/main”. Eu vou arrastar esse diretório “WebContent”, para “src/main”.

[07:23] Porém em uma aplicação Maven não se chama “WebContent”, eu vou renomeá-lo para “webapp”, tudo junto e minúsculo. Esse geralmente é o nome do diretório onde ficam os arquivos web “web.xml”, os “CSS”, “JavaScript”.

[07:40] Dentro de “WEB-INF” tem essa pasta “lib”, que era onde colocávamos os JARs da aplicação. Vou apagar essa pasta "lib" porque em uma aplicação Maven não é aqui que jogamos os JARs, configuramos isso no “pom.xml”.

[07:54] Pronto! Já migramos o projeto para seguir a estrutura do Maven. Agora, podemos alterar o arquivo “pom.xml”. Eu posso excluir aqui na *tag* `build` esse “sourceDirectory”, “testSourceDirectory”. Posso excluir esses *resources* também. OK?

[08:10] O *plugin* eu posso eu posso deixar, porque aqui o *plugin* é para trocar a versão do Java para ser o Java 11 - que é o Java que eu estou utilizando aqui na minha máquina. Esse *plugin* do WAR eu posso deixar aqui também, posso só tirar essa configuração. Não é mais “Web Content” é “Web app” padrão do Maven. Ficou assim.

[08:30] Como fizemos uma mudança aqui no Maven só para ele recarregar, dar um *clean*, atualizar o projeto, podemos clicar aqui com o botão direito do mouse em “controle-produtos" e ir em "Maven > Update Project > OK”, que ele vai atualizar o projeto para refletir essas mudanças que fizemos.

[08:48] Atualizou. Está dando erro de compilação. Se formos olhar está dando erro de compilação em todas as classes. Por quê? Por causa das bibliotecas. Nós apagamos aquela pasta “lib” e o certo agora é declararmos as dependências aqui no “pom.xml”.

[09:03] Porém, isso é assunto da próxima aula. Na próxima, aula vamos continuar aprendendo como declarar dependências. Esse projeto, para finalizar a migração dele para o Maven, bastaria adicionarmos as dependências. Porém, como ainda não vimos isso, vai ficar para o futuro.

[09:21] Então em uma próxima aula discutiremos sobre o que fazemos para declararmos as dependências de uma aplicação Maven. Vejo vocês lá, um abraço!