01 Criando um projeto com Maven

Transcrição

[00:00] Boas-vindas de volta ao treinamento de *Maven*! Continuando, agora que já conhecemos o Maven, já sabemos como é que ele funciona, vamos utiliza-lo diretamente da IDE para facilitarmos a integração que já vem, no caso, do *Eclipse*.

[00:13] Eu estou com o meu *Eclipse* aberto, em um *workspace* zerado, sem nenhum projeto. No menu do lado esquerdo, perceba que ele te dá algumas opções. Informando: você não tem nenhum projeto, quer importar um projeto, quer criar um projeto" *("There are no projects in your workspace. To add a project")*. A primeira opção que ele sugere: quer criar um projeto com Maven? *"Create a Maven project"*.

[00:30] Esse vai ser o objetivo da aula de hoje. Vamos entender como que fazemos para criar um projeto com o Maven. Você poderia clicar em "Create a Maven project" no lado esquerdo da tela ou acessar o menu "File > New > Maven Project" - vai dar na mesma, ele vai redirecionar para essa mesma tela.

[00:44] Aqui ele te pergunta se *workspace* usa esse *workspace* padrão. Tem essa primeira opção: "Create a simple project (skip archetype selection)" - em português, "criar um projeto simples (pular a escolha de um arquetipo)". Se você não marcar ela e clicar em "Next", ele vai te direcionar para essa outra tela, pedindo para você escolher um *archetype*.

[01:03] Eesse *archetype* nada mais é do que um padrão de como que ele vai criar a estrutura do projeto dependendo do objetivo do seu projeto. Ele já tem no catálogo, “Catalog”, alguns *archetypes* por padrão, que já vêm embutidos.

[01:18] Ele fala: você quer criar uma aplicação “J2ee”? Hoje em dia ninguém mais cria uma aplicação “J2ee”, mas se você quisesse criar uma aplicação “J2ee” poderia clicar nessa opção, que ele já iria criar o projeto com toda a estrutura necessária para funcionar com o “J2ee”.

[01:36] No próximo ele nos informa: você quer criar um *plugin* no Maven? - em inglês, *"org.apache.maven.archetypes maven-archetype-plugin"*. Se você quiser criar um *plugin* no Maven, você tem essa opção, que ele já cria a estrutura para um *plugin*.

[01:43] Perceba que é só uma comodidade, ele já cria o projeto com alguns arquivos e alguns diretórios baseado no tipo de aplicação que você quer construir. Em "org.apache.maven.archetypes maven-archetype-webapp", por exemplo: quer criar uma aplicação Web, uma aplicação Java Web tradicional? Ele já cria o diretório “web.info”, cria uma “web.xml” ou cria um “jsp” de exemplo. Tem essas opções disponíveis.

[02:05] Se você não quiser nenhuma delas, se quiser criar uma aplicação Maven pura, sem *archetype* nenhum e sem nada configurado; você pode voltar clicando na opção “Back > marcar o checkbox da opção "Create a simple project (skiparchetype selection)” - que cria um projeto Java com Maven sem *archetype* nenhum, só com a estrutura padrão.

[02:20] E vai desenvolvendo, à medida que vai precisando das coisas você vai criando. Quando clica em “Next” ele te pede algumas informações na janela "New Maven Project". A primeira informação que ele pede é o “*Group Id*”. *Group Id* nada mais é do que a organização, o identificador único da sua organização, da sua empresa.

[02:40] No “*Group Id*”, lembra que no Java configuramos os pacotes assim: “br.com.alura”? Geralmente esse vai ser o Group Id: "br.com.alura”. Estou identificando unicamente a empresa Alura.

[02:54] “Artifact Id” -, qual é o Id do seu projeto, do artefato. Quando você for gerar o *build* - qual é o nome desse projeto - você estará nesse grupo, “br.com.alura”, no grupo Alura, mas Alura pode ter diversos projetos e diversas aplicações. Como ele vai diferenciar uma aplicação Maven da outra pertencente a esse mesmo grupo, para isso preciso ter um Id que é único da aplicação.

[03:15] Vou colocar em “Artifact Id”, o nome "loja". Imagine que nós vamos desenvolver uma aplicação para uma loja. Aqui seria o “Artifact Id”.

[03:22] Aqui é a versão, “Version”. Qual é a versão? O Maven tem um padrão de nomenclatura para versões, você pode trocar e colocar “1.0”, usar a nomenclatura padrão. Fica a seu critério, vou colocar 1.0.0.

[03:36] Qual é o tipo de empacotamento, `Packaging`. Essa sua aplicação, quando você for gerar o *build* dela vai ser um JAR que você vai gerar, é uma aplicação Java tradicional, é uma aplicação que usa

o *spring boot*, por exemplo, que gera um JAR ou é uma aplicação web tradicional, que você vai gerar um WAR. Você já pode configurar isso. Vou deixar marcado a opção "JAR". Em `Packaging`, será "jar".

[03:56] Tem alguns campos que são opcionais, “Name” e “Description” - são só para você descrever mais o seu projeto, uma descrição do seu projeto, o nome do seu projeto. São informações opcionais.

[04:07] “Parent Project” - se o seu projeto Maven vai herdar de uma outra aplicação Maven. Depois vamos ver com calma essa situação. No caso não quero herdar de outro projeto Maven. Preenchi as informações principais e vou clicar em “Finish”.

[04:23] Ele vai fazer a criação do projeto, vai baixar algumas coisas da internet na primeira vez que você rodar. Ele criou do lado esquerdo da tela em *"Package Explorar"* a aplicação Maven, criou o projeto “loja”.

[04:33] O projeto Maven já vem sem o *archetype*, o projeto padrão puro, digamos assim; ele vem com essa estrutura.

[04:41] Tem alguns *SouceFolder* - “src/mais/java”, onde geralmente vão ficar suas classes Java. “src/main/resources”, arquivos de configuração dos seus *freameworks* e os *SouceFolder* para teste, “src/test/java” para classes de testes e “src/test/resources” para arquivos de configuração de teste.

[05:02] Embaixo vem o arquivo extremamente importante no Maven, que é o `pom.xml` - que é um arquivo de configurações do Maven para esse projeto.

[05:12] No Maven, toda a configuração é feita nesse arquivo XML - que vamos explorar ao longo do treinamento. Por padrão, o que vem preenchido no “pom”? Vem o *Group Id*, `<groupId>br.com.alura</groupId>`, conforme digitamos naquela tela de criação do projeto. *Artifact Id*, `<artifactId>loja</artifactId>`, e a versão, `<version>1.0.0</version>`.
[05:29] Esse *Model Version*, `<modelVersion>4.0.0</modelVersion>`, é o modelo, a versão do `pom.xml` do Maven, digamos assim.

[05:34] No momento de gravação do vídeo, o *Model Version* padrão atual é o 4.0. Recomendo que você utilize essa versão. Ele utiliza `http://maven.apache.org/POM/4.0.0` como *name space* do XML de configuração do “pom” na versão 4.0.

[05:50] Vou apertar as teclas “Ctrl + Shift + F” para formatar. Está aí o `pom.xml` em branco, está aí o projeto loja, as outras pastas é a estrutura de diretórios. Está vazio, não tem nada dentro de "JRE System Library", tem só as bibliotecas do próprio Java, do JRE.

[06:07] Está aí o projeto novo, criado do zero com o Maven. As pastas "src/main/java", "src/main/resources" e as outras é a estrutura de diretórios padrão do Maven e esse é o `pom.xlm` arquivo XML onde fazemos as configurações do Maven.

[06:18] Não tem muito mistério, é bem simples, bem tranquilo você criar uma aplicação do zero utilizando o Maven. A partir daí, conforme você for desenvolvendo o projeto e for precisando – “como eu adiciono as dependências, como eu mudo alguma coisa no *build*?” -

Conforme for surgindo a necessidade, você vai modificando o `pom.xml` para adicionar essas novas coisas.

[06:41] Ao longo do curso nós vamos explorando outras funcionalidades e recursos do Maven. Vejo vocês na próxima aula. Um abraço e até lá!

02 GroupId de um projeto com Maven

Vimos que ao criar um projeto com Maven foi necessário informar o **GroupId**.

Para que serve tal informação?

- Alternativa correta

  Para indicar a versão do Maven utilizada no projeto

- Alternativa correta

  Para identificar, unicamente, o nome do projeto

- Alternativa correta

  Para identificar, unicamente, a organização ao qual o projeto pertence

  Alternativa correta! O `GroupId` é como um pacote do Java, tendo como objetivo identificar a organização ao qual o projeto pertence.

03 Estrutura de um projeto com Maven

Transcrição

[00:00] No último vídeo aprendemos a criar um projeto no Eclipse utilizando o Maven. Eu até comentei por cima sobre a estrutura de diretórios, mas no vídeo de hoje eu quero focar especificamente nisso.

[00:13] Todo projeto Maven, por padrão, tem essa estrutura de diretórios.

[00:16] Ele possui esses quatro *source folders*, “src/main/java”, “src/main/resources”, “src/test/java”, “src/test/resources” e o arquivo “pom.xml”, onde ficam as configurações. Dentro de cada *source folder* desse vai determinados tipos de arquivos.

[00:34] Isso é justamente para você não ficar quebrando a cabeça, perdendo tempo pensando onde que você coloca aquele arquivo de configuração e onde que coloca o tipo de classe. Já está tudo aí pré-determinado.

[00:44] No “src/main/java”, vamos dizer assim, selecionando ele vai aparecer a tela "Select a wizard", é o principal *source folder*. É onde você vai criar as classes, interfaces, *enum's*, todo código-fonte de código Java da sua aplicação.

[00:55] Por exemplo: vamos criar uma classe digitando "class" no campo *"Wizards"*, irá abrir a janela "Java Class" e no campo `"Name"` vamos digitar "Produto" e em `"Package"` vamos colocar "br.com.alura.loja". Caminho completo: “src/main/java > Class > Next > New class > Name: 'Produto' > Package: 'br.com.alura.loja' > Finish” e a classe vai se chamar “Produto.java”, do lado esquerdo da tela.

[01:08] Em `Produto.java`, está a classe Java com os seus pacotes, classes, interfaces, toda a parte de Java mesmo fica aqui.

Código em `produto.java`:

```
package br.com.alura.loja;
```

```
public class Produto {

} COPIAR CÓDIGO
```

[01:18] Ao invés de você ficar perdendo tempo discutindo, pensando onde colocar código Java, saiba que ele deveria ficar no "src/main/java". Esse *source folder*, o objetivo dele é esse, ficar seus códigos Java.
[01:30] Se tiver páginas HTML, arquivo de configuração, não é nessa pasta que vai ficar isso. No "src/main/java" fica apenas código Java, classes, interfaces e *enums*. No geral, esses três tipos de código. No "src/main/resorces", arquivos de configuração.

[01:48] "Eu estou usando Hibernate, onde eu coloco o `persistence.xml`, o `hibernate.sfg.xml`? Estou usando o *spring build*, onde fica o `application.properties`, arquivo *Bean Validation*, `Messages.properties`, `Validation message.properties`?"

[02:02] Tudo que não é classe Java, mas que precisa estar no diretório acessível para as suas bibliotecas ou para o seu código-fonte, você coloca no "src/main/resorces".

[02:15] Selecionando "src/main/resorces", por exemplo, você poderia criar um arquivo de propriedades. Clique em " "src/main/resorces > New File > Next> no campo "File name": messages.properties > Finish". Do lado esquerdo da IDE será criado a pasta "messages.properties", arquivo de propriedades, arquivo de configurações dos seus *frameworks*, dependendo do projeto - se for uma aplicação com *Spring boot* e as páginas vão ser no *server-side*, as

páginas vão ficar dentro do “src/main/resorces”. Você cria uma pasta para colocar as páginas.

[02:39] Arquivos estáticos de CSS java script, no caso do *Spring boot* ficam dentro do “src/main/resorces”. Esses arquivos que não são código Java.

[02:48] No “src/test/java” - o nome já diz, códigos de teste. Você quer criar uma classe de teste para sua classe `Produto`? "Ctrl + N > Junit > JUnit Test Case”. Perceba que o Eclipse até detecta que a sua aplicação é uma aplicação com Maven e ele já coloca automaticamente em “source folder”: “loja/src/test/java”.

[03:06] "Se é uma classe de testes com `JUnit`, então vou colocar no "src/test/java"” E depois clicar em “Finish”. Se você criar a classe, ele vai só pedir para adicionar o JUnit do projeto - porque não tinha. Ele já cria do lado esquerdo da tela, “src/test/java”.

[03:20] Em "Produto.Test.Java" ficam as classes de testes com JUnit, TestNG ou qualquer biblioteca de testes que vocês forem utilizar no seu projeto. Fica nesse diretório.

Código em "Produto.Test.Java":

```
package br.com.alura.loja;


import static org.junit.Assert.*;


import org.junit.Test;


public class ProdutoTest {
```

```
        @Test
        public void test() {
                fail("Not yet implemented");
        }

} COPIAR CÓDIGO
```

[03:30] Se você estiver usando alguma biblioteca utilitária para facilitar os testes, *Selenium*, *Dbunit* etc., provavelmente essa biblioteca pode precisar de alguns arquivos de configuração. Como são configurações específicas para os testes, o ideal é você colocar no "src/test/resources" para não misturar com os recursos de produção, que são utilizados pela sua aplicação. Então, em "src/test/resources" é somente utilizado para os testes automatizados.

[04:00] O `pom.xml` é aquele arquivo onde ficam as configurações da sua aplicação, as configurações do *build*, das dependências e dos *plugins* que você quiser adicionar no seu projeto. Coisas que veremos ao longo do treinamento.

[04:13] Nesse vídeo era só para dar uma passada mais detalhada em cada um desses diretórios e explicar o objetivo de cada um deles. "Rodrigo, mas eu não quero usar essa estrutura de diretórios, eu quero ter o diretório "src" e o diretório "test", apenas esses dois *source folders*, posso fazer desse jeito? "

[04:31] Pode! Você pode vir no "pom.xml", tem algumas *tags* para você personalizar o *source folder* do seu projeto. Porém, não é uma boa prática e não é recomendado. Toda aplicação *Maven* segue essa

estrutura de diretórios padrão, isso é algo universal. Todo projeto, toda empresa em qualquer lugar do mundo faz desse jeito.

[04:53] Embora você consiga trocar esse padrão, o ideal é que você siga esse padrão já que isso é universal e é utilizado assim no mundo inteiro. Para que ficar reinventando a roda, se já foi definido que dessa maneira é a maneira padrão universal. Vamos seguir do jeito padrão. Não é uma boa prática você trocar essa estrutura de diretórios.

[05:12] Esse era o objetivo da aula de hoje, fazer essa discussão sobre a estrutura de diretórios de uma aplicação Maven. Vejo vocês no próximo vídeo, onde vamos aprender outros recursos do Maven. Um abraço e até lá!

04 Source folders

Um dos *sources folders* presentes em um projeto Maven é o `src/main/resources`.

Que tipos de arquivos normalmente encontramos nesse *source folder*?

- Alternativa correta

  Classe Java

- Alternativa correta

  Arquivos de configuração

  Alternativa correta! Geralmente os arquivos de configuração ficam nesse *source folder*.

- Alternativa correta

  Classes de testes automatizados

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, tem aquele projeto "loja" com aplicação Maven que criamos do zero e tem esse outro projeto 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' manualmente. 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 seria a migração - ele não faz isso automaticamente. Ele tenta descobrir analisando o código-fonte da aplicação.

[01:46] Se olharmos o "pom.xml" colocou uma *tag* `build` e em 'src ' está informando: "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 descobriu que é o diretório 'test'. É aquela configuração, '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.

Código em "controle.produtos/pom.xml":

```
//código omitido


 <plugins>
     <plugin>
     <artifactId>maven-compiler-plugin</artigactId>
     <version>3.8.0</version>
     <configuration>
         <release>11</release>
```

```
        </configuration>
    </plugin>
    <plugin>
        <artifactId>maven-war-plugin</artifactId>
        <version>3.2.1</version>
        <configuration>
            <warSourceDirectory>WebContent</warSourceDirectory>
        </configuration>
    </plugin>
</plugins>


//código omitido COPIAR CÓDIGO
```

[02:33] Já é uma aplicação Maven, porém, ela não está naquela estrutura de diretórios tradicionais 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 trabalhoso, 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 temos os dois: *source folders*, o “controle-produto/src” e “controle-produts/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 em "Add Folder" os *source folders*.

Essa é uma parte meio trabalhosa, mas se quisermos seguir a estrutura padrão, que é o recomendado, temos que seguir.

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

[03:42] Em “Add Folder > Create New Folder > digite "src/main/resources" > Finish > OK", criou o segundo. Agora, os *source folders* de teste, "Add Folder > Create New Folder > src/test/java > Finish > OK". E o último, "Add Folder > Create New Folder > src/test/resources > Finish > OK”.
[04:12] Estão aí os quatro *source folders*, porém vamos ter que fazer um ajuste no *source folder* de teste. Para dizermos certo 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) > marca a opção: "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 se adapta.

[05:16] Em “controle-produtos/src/test/resources (new)” a mesma coisa. Dois cliques em “Contains test sources: No" e no "Output folder:

(Default output folder)" também, vai abrir uma tela "Source Folder Output Location" e vamos marcar a opção "Specific output folder (path relative to 'controle-produtos')", selecionar "Browser" e escolher "target/test-classes", em seguida "OK" e "Apply".

[05:33] Nos diretórios de *resources*, no campo "Exclused (None)" está *"None"* e precisamos trocar tanto no "controle-produtos/src/main/resources" quanto no "controle-produtos/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, embaixo em "Package Explorer" eu tenho "src", dentro dele tem essa pasta "br", posso clicar nessa pasta "br" selecionar e arrastar para o "src/main/java". Pronto, já movemos o 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" 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 nela.

Ficou o “src/main” e o “src/test” e a pasta “target”. E a pasta “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 colocamos os JARs da aplicação. Vou apagar essa pasta "lib" porque em uma aplicação Maven não é nessa pasta que colocamos 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 na *tag* `build` esse `<sourceDirectory>src</sourceDirectory>` e `<testSourceDirectory>test</testSourceDirectory>`. Posso excluir esses *resources* também.

Código em `controle-produtos/pom.xml` para remover na `tag build`:

```
// código omitido


<sourceDirectory>src</sourceDirectory>
<testSourceDirectory>test</testSourceDirectory>


        <resources>
                <resource>
                    <directory>src</directory>
```

```
                    <excludes>
                        <exclude> **/*.java<exclude>
                    </excludes>
                </resource>
        </resources>


// código omitido COPIAR CÓDIGO
```

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

Código em `controle-produtos/pom.xml` para remover

```
<configuration>
    <warSourceDirectory>WebContent</warSourceDirectory>
</configuration> COPIAR CÓDIGO
```

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

[09:03] Porém, isso é assunto da próxima aula que 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!

06 Projeto sem Maven

O projeto sem Maven, mostrado no último vídeo, pode ser encontrado aqui: https://caelum-online-public.s3.amazonaws.com/2067/controle-produtos.zip

07 Faça como eu fiz

Chegou a hora de você seguir todos os passos realizados por mim durante esta aula. Caso já tenha feito, excelente. Se ainda não, é importante que você execute o que foi visto nos vídeos para poder continuar com a próxima aula.

Opinião do instrutor

Continue com os seus estudos, e se houver dúvidas, não hesite em recorrer ao nosso fórum!

08 O que aprendemos?

Nesta aula, aprendemos:

- A criar uma aplicação com Maven;
- A entender a estrutura de diretórios de uma aplicação com Maven;
- A migrar uma aplicação para o Maven.