01 Gerenciando dependências no Maven

Transcrição

[00:00] Agora que já aprendemos como fazemos para criarmos um projeto ou migrar um projeto para o Maven, chegou a hora de tratarmos de outros recursos.

[00:10] Talvez o principal recurso, a principal funcionalidade de utilização do Maven é para a parte de gerenciamento de dependências. Nesse vídeo, vamos aprender justamente como funciona essa parte de gestão de dependências com o Maven.

[00:24] Estou com aquele nosso projeto "loja", que tínhamos construído do zero. Se abrimos o "pom.xml", ele está praticamente vazio, só está com aquelas informações do `groupId`, `artifactId` e a versão do nosso projeto. Porém, não tem nenhuma dependência.

Código loja/pom.xml:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>br.com.alura</groupId>
        <artifactId>loja</artifactId>
        <version>1.0.0</version>

</project> COPIAR CÓDIGO
```

[00:39] Vamos aprender como funciona isso. Tínhamos discutido em vídeos anteriores que um dos problemas de uma aplicação Java é justamente esse de gerenciar dependências. “Eu quero usar o *Hibernate*, quero usar o *Spring MVC*, quero usar o *XStream*, quero usar o *JFreeChart* etc".

[00:57] Você tem um monte de bibliotecas para usar, você tem que baixá-las manualmente, tem que entrar no site de cada uma, fazer o download, descompactar, você tem que saber qual é a versão que você quer usar e tem que tomar cuidado com bibliotecas compartilhadas.

[01:10] Às vezes, o *JFreeChart* usa uma biblioteca qualquer, o `Log4j` na versão 1.2, mas o *Hibernate* usa na versão 1.4. E você sem querer baixou esses dois JARs e colocou junto com o projeto. Você fica com a mesma biblioteca em versões distintas e isso pode gerar um problema na sua aplicação.

[01:29] Depois, para você evoluir essa biblioteca também é trabalhoso, tem que baixar manualmente. Para livrar-se de todo esse trabalho e todo esse problema, entra o Maven com a sua principal funcionalidade de gerenciamento de dependências. Como funciona isso no Maven? É bem tranquilo. Aqui no arquivo `pom.xml` é exatamente neste arquivo que fazemos essa declaração das dependências que você quer utilizar no seu projeto.

[01:54] Aqui dentro da *tag* raiz, a *tag* `project`, pode ser embaixo da *tag* `version`, existe uma *tag* chamada `dependencies` e dentro dela você declara cada uma das pendências que você quer utilizar no seu projeto, as bibliotecas e *frameworks* que você for utilizar.

```
<dependencies>


</dependencies> COPIAR CÓDIGO
```

[02:11] Por exemplo: eu quero utilizar o `JUnit`. Ele já está na em "Junit4" do lado esquerdo da tela na aplicação, só que ele está adicionado com esse recurso do Eclipse, ele está vinculado por fora do Maven. Vamos fazer essa alteração. Eu quero tirar ele daqui e declará-lo como uma dependência.

[02:28] Vou clicar com o botão direito do mouse no projeto “loja" e selecionar "Build Path > Configure Build Path" na aba "Libraries” está o `JUnit 4`. Eu vou remover no botão do lado direito, “Remover > Apply and Close”. Perceba que começou a dar erro de compilação na classe de teste, porque ela depende do `JUnit` e eu acabei de excluí-lo do projeto.

[02:48] Só que eu vou adicioná-lo como uma dependência do Maven. Para fazer isso, lá no `pom.xml`, dentro da *tag* `dependencies`existe uma *tag* chamada ‘dependency’ e é nela que você declara uma dependência.

```
<dependencies>
    <dependency>


    </dependency>
</dependencies> COPIAR CÓDIGO
```

[03:01] Toda *tag* ‘dependency’ tem algumas coisas que você precisa informar. Por exemplo: o ‘grupoId’. Exatamente igual temos aqui o

‘groupId’ em `<groupId>br.com.alura</groupId>`. O Maven precisa saber qual é a organização que é dona dessa biblioteca, dessa dependência que você quer baixar no seu projeto.

[03:17] No caso do `JUnit`, o “groupId”, quando eles criaram a biblioteca JUnit eles colocaram aqui JUnit, simplesmente JUnit. Esse é o “groupId”.
[03:27] Qual que é o “artifactId”? Perceba que o nosso projeto pode virar uma biblioteca, uma dependência de um outro projeto. Uma dependência nada mais é do que um “groupId” e um “artifactId”. Quem é organização e quem é o projeto dessa organização que você quer utilizar como dependência. No caso do `JUnit`, o “artifactId” também se chama “JUnit”, `<artifactId>junit</artifactId>`.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"


<dependencies>
    <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
    </dependency>
</dependencies>


</project> COPIAR CÓDIGO
```

[03:50] Além dessas duas informações tem mais uma importante: existe a *tag* `version`. Qual é a versão do JUnit que você quer utilizar? No nosso projeto nós colocamos que é a versão 1.0 ,`<version>1.0.0</version>`.

[04:03] Conforme vamos evoluindo o nosso projeto, podemos ir lançando versões, tem a 1.1, a 1.2, a 1.3, a 2.0, 2.1, 2.2, 3.0 etc - você pode manter múltiplas versões de um projeto, de um `artifactId`. Se você quiser baixar uma versão específica na *tag* 'version', é onde você determina qual é a versão desse artefato.

[04:27] No caso do JUnit, eu quero usar versão 4.12, eu sei de cabeça que existe a versão 4.12.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"


<dependencies>
    <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
    </dependency>
</dependencies>


</project> COPIAR CÓDIGO
```

[04:33] Coloquei aqui a *tag* `version`, `<version>4.12</version>`. Para fechar, existe mais uma *tag* importante, chamada `scope` - que é para dizer qual é o escopo dessa dependência. "Como assim, Rodrigo, 'escopo'? " Existem alguns escopos que podemos adicionar.

[04:49] Eu utilizei o atalho "Ctrl + Espaço" e ele completou para nós. Tem o escopo de "compile", "provided", "runtime", "system", "test".

[04:56] O “compile”, você estaria dizendo: “olhe, Maven, eu preciso dessa dependência em tempo de compilação só para compilar o meu código-fonte. Quando eu for gerar o *build* do projeto você pode ignorar essa dependência. Ela só vai ser usada para compilar”. Se fosse esse o caso, você marcaria como escopo “compile”.

[05:17] Tem o “provided” que é para quando você utiliza, por exemplo, servidores de aplicação. Você precisa da dependência para compilar, para rodar o seu projeto, mas na hora de gerar o *build* você não precisa se preocupar porque ela é provida pelo próprio servidor de aplicação.

[05:32] Tem a “runtime”, que ela vai ser provida só em tempo de execução. E tem a “test”, que é o exemplo do `JUnit`. Quando você tem uma dependência que só é usada durante a execução dos testes, você não vai compilar o seu código-fonte do projeto e nem usá-lo na produção. Ela serve apenas durante a fase de rodar os testes do projeto.

[05:56] No caso do `JUnit`, é exatamente essa situação. O `JUnit` é usado apenas para rodar o teste do projeto. Existe
essa *tag* `scope`, `<scope>test</scope>` e ela tem esses tipos de escopos. Dependendo da biblioteca, você terá que dizer qual é o escopo dela para que você não a utilize em momentos errados durante o *build* e o empacotamento da sua aplicação.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"



<dependencies>
```

```
    <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
    </dependency>
</dependencies>

</project> COPIAR CÓDIGO
```

[06:17] Utilizei o atalho “Ctrl + S” para salvar. Na hora que você salvar, o Maven vai baixar essa biblioteca da internet.

[06:25] Depois vamos ver com calma de onde que ele baixa essa biblioteca. “Da internet, mas de onde?” Vamos ver com calma isso quando formos discutir sobre repositórios.

[06:33] Ele baixou a dependência e adicionou no projeto. Baixou com sucesso. Se quisermos ver se ele baixou mesmo a dependência - se expandirmos o projeto loja, perceba que embaixo de "JRE System Library" apareceu essa nova opção, “Maven Dependencies”.

[06:48] Se expandirmos “Maven Dependencies”, aparece “junit-4.12.jar”. Ele baixou o `JUnit`. Apareceu também um “hamcrest-core-1.3”, eu não declarei esse “hamcrest” no "pom.xml". Esse “hamcrest” é uma dependência do `JUnit`, uma dependência pode ter dependência.

[07:08] O `JUnit` pode usar outra biblioteca, que pode usar outra biblioteca e o Maven é que vai se virar para saber essa árvore, essa hierarquia de dependências. Eu só preciso dizer: “Maven quero

o `JUnit`. Se o `JUnit` depende de outra biblioteca, o problema é seu. Você que vai descobrir isso e você que vai baixar essa biblioteca".

[07:26] E se essa biblioteca é dependente de outra, ele quem iria sair procurando, encontrando essa hierarquia de dependências e baixando uma por uma para satisfazer a nossa dependência do JUnit no projeto.

[07:39] Perceba como é simples declarar uma dependência no Maven. É só você ir lá no seu "pom.xml", ir na *tag* `dependencies` e dentro dela adicionar a dependência. Você precisa digitar o 'groupId' ou 'artifactId', dependendo do caso você precisa da versão e dependendo do caso você precisa do escopo.

[07:58] "Poxa, Rodrigo! Mas você sabia de cabeça que esse era o 'groupId' do `JUnit`, esse era o 'artifactId' e essa era a versão. Eu vou ter que decorar isso, saber ao certo? Eu quero baixar o *Spring* no meu projeto, eu vou ter que saber qual é o 'artifactId' e o 'groupId'? " Não, você não precisa disso. Você pode pesquisar no Google, pesquisar na internet. É só copiar e colar a dependência.
[08:19] No próximo vídeo, vamos aprender exatamente a fazer isso - como pesquisar uma dependência e apenas copiar e colar no arquivo `pom.xml`. Vejo vocês lá, um abraço!

02 Pesquisando por dependências

Transcrição

[00:00] Agora que já aprendemos como fazemos para adicionarmos uma dependência no projeto - eu tinha no último vídeo, eu já sabia de cabeça qual era o “groupId” e o “artifactId” do “JUnit”.

[00:11] Mas a ideia é que você não precisa decorar esses “groupId”, “artifactId”. Sempre que precisar de uma dependência, você pode pesquisar no Google - e existe o próprio site do Maven que te ajuda. Você só precisaria copiar essa *tag* e colar dentro do seu “pom.xml”.

[00:26] Você não precisa ficar memorizando isso, perdendo tempo com isso porque é algo desnecessário. Por exemplo, vamos fazer isso: imagine que eu quero adicionar mais uma dependência nesse projeto, eu quero trabalhar com XStream - que é uma biblioteca para você fazer manipulação, conversão de Java para XML, XML para Java.

[00:44] Quero utilizar o XStream. Vou abrir o meu navegador e pesquisar no Google: “xstream maven”. É só você pesquisar dessa maneira. Um dos resultados que vai aparecer aqui no Google é esse daqui, “mvnrepository.com”. É o site do Maven neste link, onde ele tem o repositório, onde está declarado todas as dependências.

[01:06] Nesse site tem a dependência do XStream e vai aparecer todas as versões. Você precisa escolher qual é a versão a ser utilizada.

[01:16] Talvez você queira usar uma determinada versão, bastaria você clicar nela - ou se você não tem uma versão específica, o ideal é pegar a última versão estável. No caso seria essa, `1.4.14`, que foi lançada em novembro de 2020, há pouco tempo. Quero usar essa versão. Você clica na versão e ele te mostra exatamente a *tag*, no caso do Maven,

‘dependency’. Bastaria você dar um “Ctrl + C” nessa *tag* e colar lá no seu “pom.xml”.

[01:44] Ele também suporta para outras bibliotecas de gestão de dependências, como *Ivy*, *SBT* e *Gradle*. No nosso caso é o Maven.

[01:54] Eu copiei essa *tag*, vou no Eclipse, embaixo da dependência do “JUnit” em `loja/pom.xml` e vou colar. Vou só usar o atalho “Ctrl + Shift + F” para ele formatar. E está aí, colei! Vou salvar e nesse momento ele vai baixar da internet.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"


//código omitido


<dependencies>
    <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
    </dependency>
            <dependency>
            <groupId>com.thoughtworks.xstream</groupId>
            <artifactId>xstream</artifactId>
            <version>1.4.14</version>
        </dependency>
</dependencies>


</project>
```

```
COPIAR CÓDIGO
```

[02:08] Se for a primeira vez que você está baixando essa dependência, está declarando essa dependência nesse computador, o Maven vai baixar da internet a dependência do XStream, todas as subdependências dessa biblioteca e salvar no diretório local da sua máquina. Depois eu mostro para vocês esse diretório. Se ele já tivesse baixado anteriormente, ele buscaria do seu repositório local que fica dentro do seu computador.

[02:33] Ele baixou! Perceba, vamos analisar no projeto, naquele “Maven Dependencies”, só tinha o “JUnit e o “hamcrest”. Agora, perceba que ele baixou o "xstream-1.4.14.jar" e mais duas dependências, que são dependências do XStream, o “xmlpull” e o “xpp3”.

[02:53] Não sei o que é isso, é dependência do XStream. Para mim não interessa. Eu quero o XStream, se ele depende dessas bibliotecas, é problema dele, é problema do Maven. O Maven é quem vai descobrir e baixar isso.

[03:04] Perceba, para você adicionar uma dependência no seu projeto é só você pesquisar no Google o nome da biblioteca + tecla "Espaço" + "Maven”, que vai cair no Maven Repository. Você escolhe a versão, copia a *tag* ‘dependency’, cola no seu “pom.xml”. Você não precisa ficar memorizando esse “groupId”, esse “artifactId”. Essa versão isso tudo já está disponibilizado para você automaticamente no site do Maven.

[03:31] Espero que vocês tenham aprendido e tenham gostado desse jeito mais prático. Vejo vocês no próximo vídeo. Um abraço e até lá!

03 Alterando o repositório remoto de dependências

Transcrição

[00:00] Agora que já vimos como adicionar uma dependência no Maven através da *tag* `dependency` lá no arquivo "pom.xml" e também a pesquisar pela dependência para não ter que ficar memorizando o "groupId", "artifactId".

[00:13] Chegou a hora de entendermos melhor como o Maven baixa essas dependências. Se analisarmos o `pom.xml`, em nenhum local dissemos de onde que o Maven vai baixar essas dependências, se é da internet ou do computador.

[00:28] Isso não foi configurado porque, por padrão, o Maven vai primeiro buscar no repositório local que está no seu computador. Daqui a pouco eu mostro esse repositório para vocês. E se ele não encontrar essa dependência no repositório local, ele pesquisa na internet em um repositório central do próprio Maven.

[00:45] Se você quiser conhecer um pouco sobre esse repositório central, você pode abrir o Google e pesquisar "Maven repository" e vai aparecer esse site, mvnrepository.com neste link. Esse é o repositório central do Maven.

[01:02] É onde estão todos os projetos, os *frameworks*, as bibliotecas e com tudo disponibilizado nesse repositório central do Maven. É que ele faz a consulta e faz o download do JAR e das dependências conforme o seu projeto demandar. Esse é o repositório central.

[01:18] Porém, pode acontecer de determinada dependência que você quer utilizar no seu projeto não estar disponibilizada no repositório central no Maven e estar disponibilizado em um outro repositório de outra empresa, ou pode ser até um repositório privado, interno da sua organização.

[01:36] Você quer manter um controle melhor, não quer que o pessoal da área de programação baixe diretamente da internet. Você quer deixar tudo local na sua empresa. Tem como você configurar um repositório interno da sua empresa ou utilizar um outro repositório que não seja o do próprio Maven.

[01:52] Para fazer isso é só você vir no arquivo `pom.xml`. Dentro da *tag* `project`, da *tag* raiz do `pom.xml`, você adiciona uma *tag* chamada "repositories". Eu já tenho ela copiada, só para não perdermos muito tempo. Vou colar e essa é a *tag* `<repositories>`. Aqui dentro você pode adicionar um ou mais repositórios, que o Maven sempre vai procurar a ordem de declaração.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"

//código omitido

    <repositories>
        <repository>
```

```
            <id>spring-repo</id>
            <url>https://repo.spring.io/release</url>
        </repository>
    </repositories>
</project> COPIAR CÓDIGO
```

[02:17] Tem a *tag* `<repository>`, tem o `<id>`. Aqui, por exemplo, eu coloquei o repositório do *Spring*. O *Spring* tem um repositório próprio que tem lá os JAR, as bibliotecas do *Spring*.

[02:27] Tem o `<id>` e uma `<url>`, o que interessa mesmo é só a URL, `<url>https://repo.spring.io/release</url>`. A partir de agora quando eu solicitar para o Maven adicionar uma nova dependência no projeto. Ele vai pesquisar nesse repositório, um novo repositório remoto que você acabou de adicionar.

[02:41] Em `<url>https://repo.spring.io/release</url>` não precisa necessariamente ser um repositório hospedado na internet. Ao invés de "http" poderia colocar "file:///home/", uma pasta da minha máquina, ou poderia ser "http://ip" de um servidor interno da sua empresa. Pode ser algo interno, algo local no seu computador ou algo hospedado na internet - seja um repositório no Maven ou de outros projetos.

[03:16] Tem o repositório do *Spring*, tem o repositório também da *JBoss*, tem alguns outros repositórios que você pode utilizar. Vou deixar esse salvo aqui de exemplo, `<url>https://repo.spring.io/release</url>`.

[03:25] Toda vez que você adiciona uma nova dependência no `pom.xml`, o Maven primeiramente vai olhar no *cache* local da sua máquina para ver se já existe essa dependência salva no seu computador, porque aí ele não precisa acessar nenhum endereço para baixar essas dependências, então acaba sendo mais rápido.

[03:43] Só se ele não encontrar no *cache* local é que ele vai acessar os repositórios que você configurou ou acessar o repositório central do Maven. Esse repositório local, esse *cache*, é um diretório chamado ".m2".

[03:58] Como a pasta começa com "." é um diretório oculto. Ele fica localizado no diretório "Home" do seu usuário no seu computador. Isso vai depender do sistema operacional. Eu estou usando o Linux. Eu estou dentro do meu diretório "Home", ele só está mostrando as pastas comuns. Se eu quiser exibir as pastas ocultas eu utilizo o atalho "Ctrl + H" - e perceba que tem uma pasta chamada ".m2".

[04:24] Esse é o *cache* local do Maven. Se entrarmos na pasta ".m2" tem uma pasta chamada "repository" e dentro dessa pasta tem todos os JARs, todas as bibliotecas que ele já baixou.

[04:34] Percebe que tem muitas pastas, porque são as dependências, as dependências das dependências e aí vai de maneira hierárquica. Por exemplo: baixamos o `JUnit`, tem uma pasta aqui "JUnit" e dentro dela tem as versões do "JUnit". Já tinham algumas de outros projetos desse computador. A "4.12" foi a que baixámos, está "Junit-4.12.jar". Tem alguns arquivos específicos que o Maven utiliza.

[05:02] Esse é o repositório local. Toda vez que você adiciona uma nova dependência no projeto, primeiro o Maven analisa na sua pasta ".m2" para ver se ele encontra essa dependência. Se não encontrar, ele pesquisa na internet no repositório central dele ou nos repositórios que você configurar no `pom.xml` do seu projeto.

[05:20] Uma dica importante: às vezes acontece um determinado problema. Você adiciona uma dependência no `pom.xml`, copia ela da internet e ela está certa só que está dando algum erro. Ele fica marcando de vermelho e você olha e está tudo certo, a dependência está certa, o "groupId" está certo e "artifactId" está certo.
[05:38] Às vezes acontece algum problema na hora que o Maven baixa essa dependência da internet. Se tiver um erro e ele não se resolver. Se você clicar com o botão direito do mouse no projeto em "loja > Maven > Update Project > OK", o Maven vai meio que sincronizar. Mesmo assim, caso o erro continue. Às vezes, pode ser um problema do seu repositório local, do seu *cache* que deu algum problema na hora de baixar uma dependência.

[06:06] Uma dica que eu dou é quando isso acontecer entra na pasta ".m2 > repository > seleciona tudo > Shift + Delete", apague tudo. Depois que ele concluir a exclusão, você vai no Eclipse e faz aquele "Update Project" novamente: "loja > Maven > Update Project > OK". Isso vai forçar o Maven a baixar as dependências.

[06:31] Perceba que embaixo, eu vou clicar no ícone no canto inferior do lado direito da tela para ver a aba "Progress". Agora ele está atualizando o projeto.

[06:39] Ele não achou no repositório local na minha máquina, por isso ele está baixando da internet. Quando ele vai baixar da internet, geralmente demora um pouco. Vai depender da sua conexão, do número de dependências que você tem no seu projeto e ele vai baixando cada uma dessas dependências e vai criando os diretórios.

[06:55] Os diretórios são de acordo com o “artifactId” e o “groupId”. Se for um nome no "groupId" de “com.alura.projeto” ele vai criando esses diretórios como se fossem pacotes do Java. Ele vai criando os diretórios e baixando os JARs, as dependências.

[07:14] Tem coisas que ele baixa que não estão no nosso projeto, que são *plugins* e coisas internas do próprio Maven. Não necessariamente ele só vai baixar essas duas dependências que constam em `pom.xml`, ele vai baixar também as dependências de cada uma dessas dependências e *plugins* e coisas internas do próprio Maven - por isso que está demorando e provavelmente vai demorar alguns minutos.

[07:32] Esse era o objetivo do vídeo de hoje, explicar para vocês essa questão do repositório - que o Maven tem esse *cache* local no seu computador, essa pasta “.m2”, que fica no diretório “Home” do seu usuário.

[07:44] Dependendo do sistema operacional, é de um jeito diferente para você acessar este diretório. Se ele não encontrar lá, ele busca na internet, no repositório central do Maven, no site mvnrepository.com, ou no seu `pom.xml`. Se você tiver configurado algum repositório, ele vai buscar por lá.

Código pom.xml:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>br.com.alura</groupId>
    <artifactId>loja</artifactId>
    <version>1.0.0</version>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>com.thoughtworks.xstream</groupId>
            <artifactId>xstream</artifactId>
            <version>1.4.14</version>
        </dependency>
    </dependencies>

    <repositories>
        <repository>
            <id>spring-repo</id>
            <url>https://repo.spring.io/release</url>
        </repository>
```

```
    </repositories>
</project> COPIAR CÓDIGO
```

[08:03] É assim que funciona os repositórios e você pode ter um repositório privado, interno da sua empresa, se você quiser filtrar e limitar as bibliotecas e dependências que a sua equipe e o time de desenvolvimento da sua empresa, pode baixar. Você pode utilizar um repositório privado e pode usar também uma ferramenta mais avançada para gerenciar as dependências e artefatos, como *Nexus*, dentre outras. Mas esse não será o foco desse treinamento.

[08:28] Espero que vocês tenham gostado. Vejo vocês no próximo vídeo onde continuaremos aprendendo outros recursos do Maven. Um abraço!

04 Repositório central

Por padrão, de onde o Maven baixa as dependências de uma aplicação?

- Alternativa correta

  Do repositório central do próprio Maven

  Alternativa correta! O Maven possui um repositório central de dependências.

- Alternativa correta

  Precisamos configurar manualmente o repositório de onde ele vai baixar as dependências

- Alternativa correta

  As dependências já vem instaladas juntamente com o Java

05 Repositórios remotos

Vimos que é possível adicionar outros repositórios remotos para o Maven baixar as dependências de uma aplicação.

Por qual motivo precisaríamos adicionar algum repositório remoto?

- Alternativa correta

  Para ter um repositório de backup

- Alternativa correta

  Para que o Maven baixe mais rapidamente as dependências

- Alternativa correta

  Para baixar dependências que não estão no repositório central do Maven

  Alternativa correta! Pode acontecer de alguma dependência da aplicação ser disponibilizada apenas em outros repositórios remotos.

06 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!

07 O que aprendemos?

Nesta aula, aprendemos:

- Como declarar as dependências de uma aplicação;
- Como pesquisar por dependências no repositório central do Maven;
- Como configurar outros repositórios remotos.