

HIBERNATE

PARTE 2



JPA - Hibernate x Maven (Apache)

2

- Maven - ferramenta de gerenciamento de dependências e do ciclo de vida de projetos:
 - Facilitar a compilação do código, o empacotamento (JAR, WAR, EAR, ...), a execução de testes unitários, etc.
 - Unificar e automatizar o processo de geração do sistema. Nada mais de uma coleção de passos e scripts a serem executados manualmente.
 - Centralizar informações organizadas do projeto, incluindo suas dependências, resultados de testes, documentação, etc.
 - Reforçar boas práticas de desenvolvimento, tais como: separação de classes de teste das classes do sistema, uso de convenções de nomes e diretórios, etc.
 - Ajuda no controle das versões geradas (*releases*) dos seus projetos.

Maven - Alguns Conceitos

3

Artefato (artifact)

- ▣ Um artefato é geralmente um arquivo JAR que fica no repositório do Maven, mas pode ser de outro tipo.
- ▣ Cada artefato é identificado através dos seguintes elementos:
 - **Grupo**: é como o endereço do site ao contrário: br.com.programarcomvoce
 - **Identificador único de artefato**: geralmente é o nome do projeto. Ele deve ser único por grupo.
 - **Número da versão**: a versão do projeto, como 1.4.2. Se houver o sufixo **-SNAPSHOT**, (**0.0.1-SNAPSHOT**, por exemplo) significa que o projeto está em desenvolvimento e o pacote pode ser alterado.
 - **Tipo do projeto**: jar, war, ear, pom (projeto de configuração).
 - **Classificador**: identificação opcional para diferenciar variações da mesma versão. Por exemplo, se o programa é compilado para diferentes versões do Java podemos usar os classificadores jdk4 e jdk6. e há variações específicas para Sistemas Operacionais, podemos ter os classificadores **linux** e **windows**.

Maven - Alguns Conceitos

4

Repositório Local

- É um diretório em seu PC onde os artefatos são armazenados após baixados de um repositório remoto na internet ou na intranet. Você também pode instalar os artefatos dos seus projetos nesse repositório executando o *install* do Maven.
- O repositório possui uma estrutura padrão onde o Maven consegue encontrar os artefatos através da identificação do mesmo.

Maven - Alguns Conceitos

5

Repositório remoto

- ❑ Consiste numa aplicação que disponibiliza artefatos do Maven. Pode se um repositório público na Internet, onde criadores de bibliotecas e frameworks disponibilizam seus artefatos, ou pode ser um repositório privado da empresa, disponível na intranet.
- ❑ Existe um repositório central que já vem configurando no Maven, mas algumas empresas criam seus próprios repositórios. Inclusive você pode criar o seu instalando o *Artifactory* ou *Nexus* num servidor.
- ❑ Quando adicionamos esses repositórios remotos em nossa instalação do Maven, ele é capaz de localizar e baixar automaticamente as dependências através da identificação do artefato.

Maven - Alguns Conceitos

6

Arquivo POM

- O arquivo pom (*pom.xml*) é a configuração principal do Maven e estará presente em todo projeto. Nele você declara a identificação do seu projeto (que após gerado será também um artefato Maven), as dependências, repositórios adicionais, etc.
- Há um arquivo pom por projeto, mas ele pode herdar configurações de um parent pom, isto é, como se houvesse um projeto "pai".

Maven - Alguns Conceitos

7

Ciclo de vida padrão do Maven

- *Validate*: valida o projeto e se as informações necessárias para os próximos passos estão disponíveis, como as dependências por exemplo.
- *compile*: compila o código-fonte.
- *test*: executa os testes unitários (JUnit, por exemplo).
- *package*: empacota o código compilado em um JAR, WAR, etc.
- *Integration-test*: executa os testes de integração.
- *install*: adiciona o pacote gerado ao repositório local, assim outros projetos na mesma máquina podem usar essa dependência.
- *deploy*: copia o pacote final para o repositório remoto, disponibilizando-o para outros desenvolvedores e outros projetos.

No Maven, você pode configurar detalhadamente cada um desses passos e até incluir novos.

Por exemplo, enquanto você está desenvolvendo um módulo, a cada alteração pode executar o passo test para executar a validação, compilação e então os testes unitários. Então você só precisa executar os passos posteriores quando tiver concluído o trabalho.

Maven - Alguns Conceitos

8

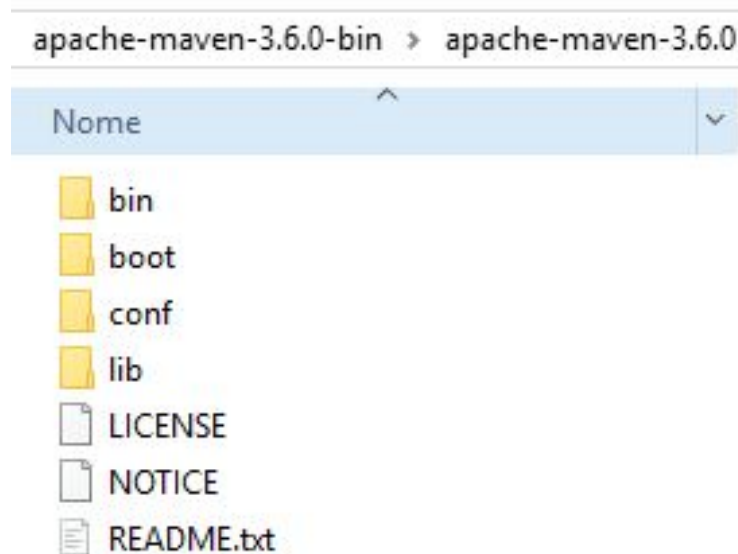
Estrutura padrão de um projeto Maven

- A estrutura padrão do projeto inclui boas práticas (como separar as classes de teste das classes do sistema) e facilita aos novos desenvolvedores encontrar o que eles querem, já que todos os projetos seguirão uma estrutura semelhante.
 - *src/main/java*: aqui fica o código-fonte do sistema ou biblioteca.
 - *src/main/resources*: arquivos auxiliares do sistema, como *.properties*, XMLs e configurações.
 - *src/main/webapp*: se for uma aplicação web, os arquivos JSP, HTML, JavaScript CSS vão aqui, incluindo o *web.xml*
 - *src/test/java*: as classes com seus testes unitários ficam aqui e são executadas automaticamente com JUnit e TestNG. Outros frameworks podem exigir configuração adicional.
 - *src/test/resources*: arquivos auxiliares usados nos testes. Você pode ter properties e configurações alternativas, por exemplo.
 - *pom.xml*: é o arquivo que concentra as informações do seu projeto.
 - *target*: é o diretório onde fica tudo que é gerado, isto é, onde vão parar os arquivos compilados, JARs, WARs, JavaDoc, etc.

Instalando o Maven (Apache)

9

- Baixar o Drive na pasta “INSTALADORES” o arquivo “apache-maven-3.6.0-bin.zip”;
- Descompactar o arquivo em uma pasta a sua escolha ou dentro da pasta de seus projetos java;



Configurando o Maven (Apache)

10

- O Maven é configurado através do arquivo *settings.xml* que fica no diretório *... \conf*.
- Em um ambiente simples você não vai precisar mexer em muita coisa. Porém, existem alguns pontos pontos mais importantes, como *proxy*, e o local do repositório (local para baixar os artefatos da internet. O diretório padrão fica da pasta do usuário, na pasta *.m2*.

Configurando as variáveis de ambiente

11

- Vá nas “Variáveis de Ambiente” do windows crie uma nova variável chamada “*M2_HOME*” e informe o caminho que você descompactou o maven até a pasta “*bin*”.. Exemplo:
“*E:\CursoPCV\apache-maven-3.6.0-bin\apache-maven-3.6.0\bin*”:
- Agora incluir o diretório com os executáveis do Maven ao *PATH*. Localize a variável, clique em Editar... e adicione ao final um ponto e vírgula e o caminho para a pasta bin do Maven
(*;%M2_HOME%\bin*)

Testando Instalação Maven

12

- Abra o **CMD** (linha de comando) e digite:
mvn --version
- Deve aparecer como a figura abaixo:



```
Prompt de Comando
Microsoft Windows [versão 10.0.17134.648]
(c) 2018 Microsoft Corporation. Todos os direitos reservados.

C:\Users\altca>mvn --version
Apache Maven 3.6.0 (97c98ec64a1fdfee7767ce5fffb20918da4f719f3; 2018-10-24T15:41:47-03:00)
Maven home: E:\CursoPCV\apache-maven-3.6.0-bin\apache-maven-3.6.0\bin\..
Java version: 1.8.0_201, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jre1.8.0_201
Default locale: pt_BR, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"

C:\Users\altca>
```

- Se ocorreu algum problema, verifique se você tem o Java instalado e configurado corretamente, incluindo as variáveis de ambiente

Usando o Maven - Eclipse

13

- Para integrar o Maven e Eclipse, usaremos o plugin M2E.
- A distribuição *Eclipse for JEE Developers* já vem com o plugin M2E e uma instalação interna do Maven.
- Acesse o menu *Window > Preferences...* e vá até a opção *Maven > Installations*.

Usando o Maven - Eclipse

14

- Veja que já existe uma instalação “embarcada”, mas com uma versão desatualizada. Vamos adicionar o nosso Maven que baixamos:
- Clique em Add... E selecione a pasta com a sua instalação do Maven, no meu caso:

E:\CursoPCV\apache-maven-3.6.0-bin\apache-maven-3.6.0

Criando um Projeto no Eclipse

15

- Crie um novo projeto do tipo “Maven Project”;
- Na primeira tela, selecione a opção “Create a simple project (skip archetype selection)”, clique em *Next...*
- Na segunda tela, vamos preencher a identificação do projeto, que nada mais é do que a identificação de um artefato:
 - Group Id: br.com.cursopcv
 - Artifact Id: teste-maven-01
 - Version: 0.0.1-SNAPSHOT
 - Packaging: jar
 - Name: Teste Maven 01
 - Description: Um teste de projeto simples com o maven
- O *Group Id* para o exemplo será *br.com.cursopcv* e o *Artifact Id* será teste-maven-01. A versão e o tipo de artefato (Packaging) já devem estar preenchidos, então simplesmente deixe como está. O nome e a descrição são opcionais. Clique em *Finish* para ver o projeto criado.

Codificando o Encoding para UTF-8

16

- No Eclipse, abra o arquivo *pom.xml*. Adicione a tag *<properties>* logo abaixo da tag *<version>* e cole o conteúdo abaixo descrito:

```
<properties>
```

```
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
```

```
</properties>
```


Adicionando Manualmente uma dependência

17

- ❑ Acesse o site mvnrepository.com, que contém um índice das dependências disponíveis no repositório do Maven. Pesquise por **commons-lang**.
- ❑ Selecione o item *Apache Commons Lang*;
- ❑ Clique sobre a última versão (**3.8.1**);
- ❑ Vai aparecer informações da <dependency>, desmarque a opção “Include comment with link to declaration”, marque todas as linhas e copie.

Adicionando Manualmente uma dependência

18

- Agora, no Eclipse, abra o arquivo *pom.xml*. Adicione a tag *<dependencies>* logo abaixo da tag *<version>* e cole o conteúdo copiado dentro dela.
- Pressione *CTRL+A* para selecionar todo o conteúdo do arquivo e depois *CTRL+I* para indentar o arquivo;
- Salve o arquivo. O plugin M2E irá identificar a alteração, baixar automaticamente a dependência do repositório central para o seu repositório local e adicioná-la ao *classpath* do projeto.

Testando as classes do Apache Commons Lang

19

- Agora, podemos usar qualquer classe da biblioteca *Apache Commons Lang*.
- Dentro da *package src/main/java* crie uma nova package “*br.com.cursopcv.testemaven*”;
- Crie uma classe chamada *ClasseDeTesteData*;

Testando as classes do Apache Commons Lang

20

- A biblioteca *Apache Commons Lang*, oferece diversas funcionalidades utilitárias com métodos para trabalhar com arrays ou strings, por exemplo. Vale a pena estudar as classes como:
 - StringUtils
 - StringEscapeUtils
 - ArrayUtils
 - EqualsBuilder e HashcodeBuilder
 - ToStringBuilder
 - NumberRange
 - DateUtils
- No nosso exemplo, vamos utilizar um exemplo da classe *DateUtils*

Testando as classes do Apache Commons Lang

21

- Na classe “*ClasseDeTeste*”, vamos fazer uma manipulação de datas. Poderíamos usar a classe *Calendar*, mas vamos usar a classe *DateUtils* que abstrai o uso do *Calendar*. Como no exemplo:

.....

```
Date hoje = new Date();
```

```
Date proxSemana = DateUtils.addDays(hoje, 7);
```

```
SimpleDateFormat dtformato = new SimpleDateFormat("dd/MM/yyyy");
```

```
String hojeFormatada = dtformato.format(hoje);
```

```
String proxSemanaFormatada = dtformato.format(proxSemana);
```

```
System.out.println("Data de Hoje.....: " + hojeFormatada);
```

```
System.out.println("De hoje a 7 dias: " + proxSemanaFormatada);
```

.....

Executando os passos (goals) do Maven

22

Ciclo de vida padrão do Maven

- *Validate*: valida o projeto e se as informações necessárias para os próximos passos estão disponíveis, como as dependências por exemplo.
- *compile*: compila o código-fonte.
- *test*: executa os testes unitários (JUnit, por exemplo).
- *package*: empacota o código compilado em um JAR, WAR, etc.
- *Integration-test*: executa os testes de integração.
- *install*: adiciona o pacote gerado ao repositório local, assim outros projetos na mesma máquina podem usar essa dependência.
- *deploy*: copia o pacote final para o repositório remoto, disponibilizando-o para outros desenvolvedores e outros projetos.

□ Vamos construir uma nova biblioteca. Precisaremos testá-la (*test*), empacotá-la (*package*) num jar e distribuí-la para uso de terceiros. O Maven nos ajuda grandemente com esses passos naturais do ciclo de vida de um projeto.

Executando os passos (goals) do Maven

23

- Vamos construir uma nova biblioteca. Precisaremos testá-la (*test*), empacotá-la (*package*) num jar e distribuí-la para uso de terceiros. O Maven nos ajuda grandemente com esses passos naturais do ciclo de vida de um projeto.
- Crie uma classe chamada “*TempoExecucaoCodigo*” dentro do pacote *br.com.cursopcv.testemaven*;
- Nesta classe criar um método para calcular o tempo de execução do nosso código do exercício anterior;

Executando os passos (goals) do Maven

24

```
public static long tempoExecucaoPrograma() {  
    long tempo = 0;  
    Stopwatch sw = new Stopwatch();  
    sw.start();  
  
    Date hoje = new Date();  
    Date proxSemana = DateUtils.addDays(hoje, 7);  
    SimpleDateFormat dtformato = new SimpleDateFormat("dd/MM/yyyy");  
    String hojeFormatada = dtformato.format(hoje);  
    String proxSemanaFormatada = dtformato.format(proxSemana);  
  
    sw.stop();  
  
    return tempo = sw.getTime();  
}
```


Executando os passos (goals) do Maven

25

```
public static long tempoExecucaoPrograma() {  
    long tempo = 0;  
    Stopwatch sw = new Stopwatch();  
    sw.start();  
  
    Date hoje = new Date();  
    Date proxSemana = DateUtils.addDays(hoje, 7);  
    SimpleDateFormat dtformato = new SimpleDateFormat("dd/MM/yyyy");  
    String hojeFormatada = dtformato.format(hoje);  
    String proxSemanaFormatada = dtformato.format(proxSemana);  
  
    sw.stop();  
  
    return tempo = sw.getTime();  
}
```

Executando os passos (goals) do Maven

26

- Vamos criar um teste unitário para a nossa classe criada. Para isso, vamos primeiro adicionar a dependência do JUnit ao nosso projeto.
- Com o Junit podemos criar testes para verificar funcionalidades de classes e seus métodos.
- Para isso, vá até o site mvnrepository.com e pesquise por [junit](#). Vá até a [última versão](#), copie o trecho do XML e adicione na seção de dependências do seu [pom.xml](#).

```
<dependencies>
  <dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
    <version>3.8.1</version>
  </dependency>

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

</dependencies>
```

Executando os passos (goals) do Maven

27

- Vamos criar a classe de teste “*ClassTestCalcTempo*” para chamar-mos o método *tempoExecucaoPrograma()*;

```
@Test
```

```
public void medirTempo() {
```

```
    long tempo = TempoExecucaoCodigo.tempoExecucaoPrograma();
```

```
    System.out.println("Tempo gasto para executar o trecho do programa: " + tempo + " milissegundos");
```

```
}
```

- Para executar o teste, sobre a classe clique com o botão direito e acesso o menu *Run As > Junit Test*.
- Com isso, temos uma versão beta de nossa biblioteca;
- Vamos testar o projeto usando o Maven. Mude para a perspectiva java e clique no projeto com o botão direito e na opção *Run As > Maven test*;

Executando os passos (goals) do Maven

28

```
<terminated> C:\Program Files\Java\jre1.8.0_201\bin\javaw.exe (26 de mar de 2019 15:11:46)
[INFO] Scanning for projects...
[INFO]
[INFO] -----< br.com.cursorpcv.frameworks:teste-maven-01 >-----
[INFO] Building Teste Maven 01 0.0.1-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ teste-maven-01 ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ teste-maven-01 ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ teste-maven-01 ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ teste-maven-01 ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ teste-maven-01 ---
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.940 s
[INFO] Finished at: 2019-03-26T15:11:49-03:00
[INFO] -----
```

Gerar o JAR do projeto

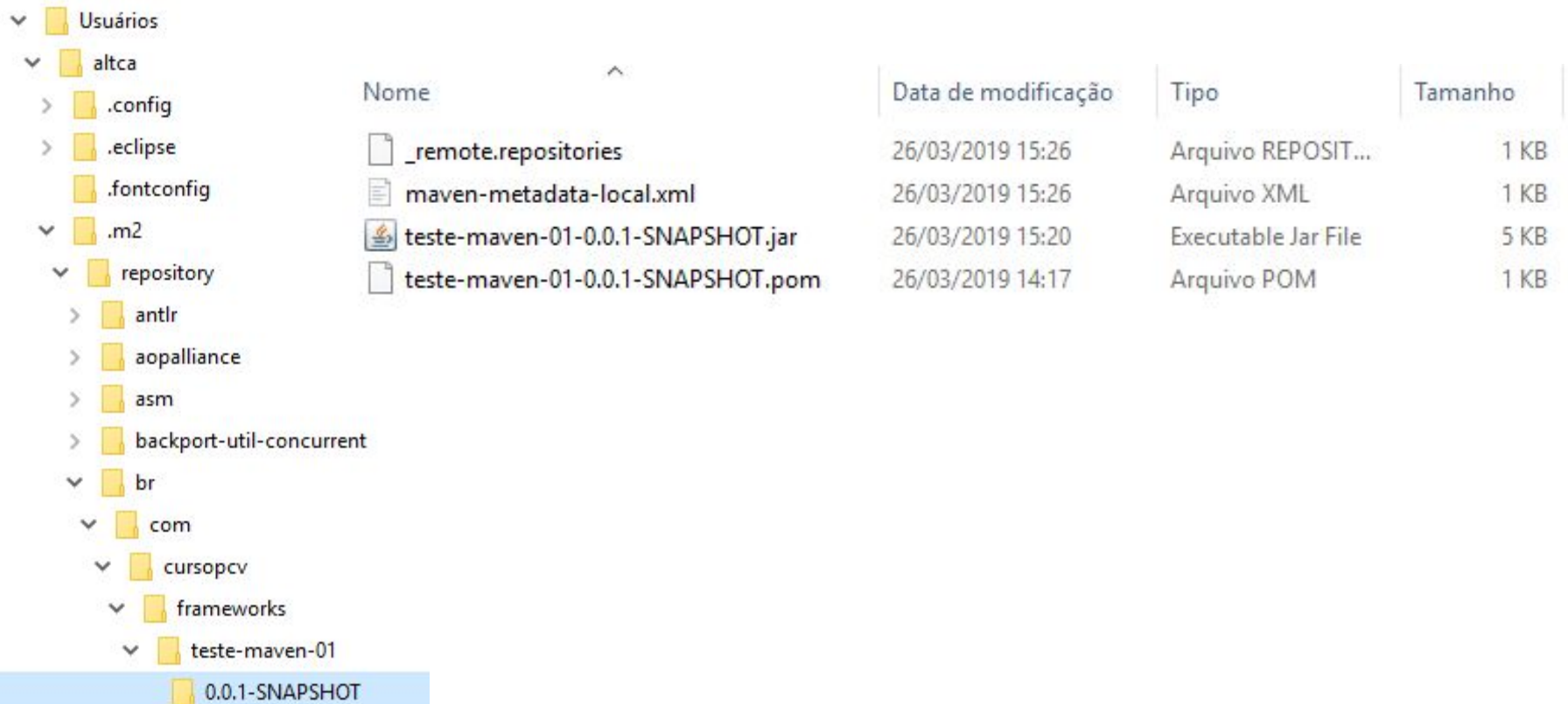
29

- Com os testes feitos com sucesso! Vamos gerar um JAR do projeto.
- Clique no projeto com o botão direito e na opção *Run As > Maven build...*. Na tela que abrir, vá até o campo *Goals* e digite *package*. Clique em *Run* e aguarde;
- Observe o log no console e notará que os testes foram executados. Todos os passos foram executados.
- Note a última linha antes das palavras *BUILD SUCESS*. Ali está o caminho do JAR gerado. Ele foi gerado na pasta *target* do projeto.
- Selecione a pasta *target* e pressione *F5* para atualizá-la.

Instalar o jar no repositório local

30

- Clique no projeto com o botão direito e na opção *Run As > Maven install*. Aguarde um pouco.....
- Abra o console e veja que as duas últimas linhas antes da mensagem de sucesso demonstram os locais onde o Jar e o seu arquivo POM foram instalados.



Nome	Data de modificação	Tipo	Tamanho
_remote.repositories	26/03/2019 15:26	Arquivo REPOSIT...	1 KB
maven-metadata-local.xml	26/03/2019 15:26	Arquivo XML	1 KB
teste-maven-01-0.0.1-SNAPSHOT.jar	26/03/2019 15:20	Executable Jar File	5 KB
teste-maven-01-0.0.1-SNAPSHOT.pom	26/03/2019 14:17	Arquivo POM	1 KB

Instalar o jar no repositório local

31

- Agora você pode usar este artefato em outros projetos na sua máquina local, adicionando a seguinte dependência:

```
<dependency>
  <groupId>br.com.starcode</groupId>
  <artifactId>teste-maven-01</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</dependency>
```

- Qualquer projeto com essa dependência vai usar o nosso jar gerado e, automaticamente, incluir também o jar do Apache Commons Lang que definimos em nosso projeto.
- Os próximos passos incluiriam disponibilizar o jar para outros desenvolvedores através do goal *deploy*. Em um deploy, o Maven envia seu jar para um Repositório Remoto. Entretanto, isso exige várias configurações adicionais e as devidas permissões.

Exercício de Fixação