

Linguagem de Programação

Introdução ao Apache Maven

O que é o Apache Maven

Toda plataforma de desenvolvimento de software moderna permite a criação e importação de **bibliotecas**. Elas são, basicamente, projetos que são incorporados ao nosso, permitindo o reaproveitamento de funcionalidades. Essas bibliotecas podem ser de código aberto ou não, bem como pagas ou gratuitas.

Na plataforma Java, uma biblioteca é um arquivo com a extensão **.jar**. Basicamente, é um arquivo compactado com o mesmo algoritmo dos **.zip** que contém vários arquivos **.class** (arquivos compilados para a JVM). Esses arquivos são encontrados facilmente na internet, em se tratando de bibliotecas gratuitas, claro.

Porém, em projetos do mundo real, raramente precisamos de apenas uma biblioteca. E mais, cada biblioteca provavelmente usa outra ou outras. E essas outras, também usam mais bibliotecas. Enfim, é comum acabarmos precisando, indiretamente, da biblioteca da biblioteca da biblioteca... Pense no trabalho que daria ficar procurando uma a uma na internet e depois instalar cada uma delas em seu projeto. Um exemplo disso seria se precisássemos de uma biblioteca chamada **spring-boot-starter**, versão **2.1.3.RELEASE**. Veja na Figura 1 como seria apenas uma parte de sua hierarquia de dependências.

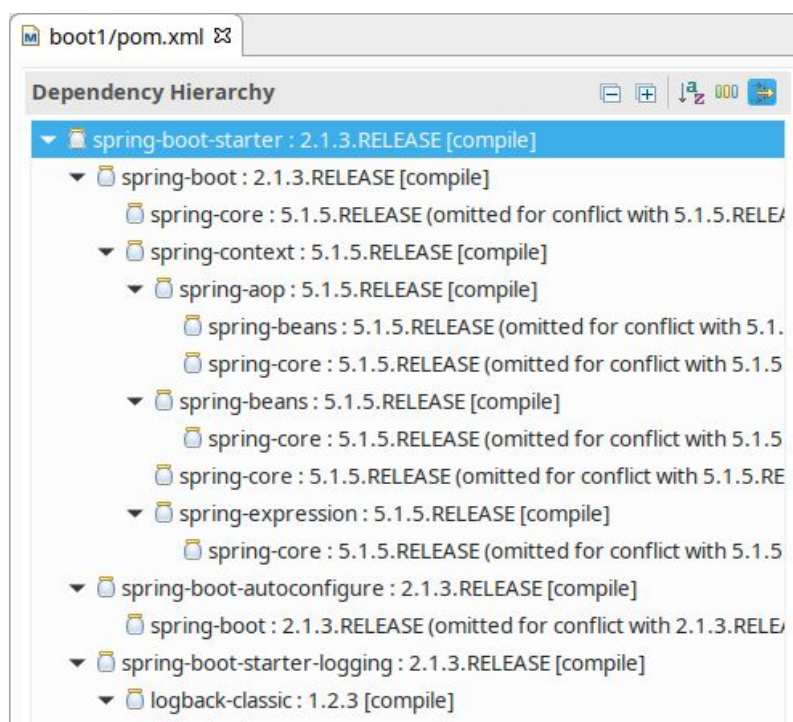


Figura 1: Parte da hierarquia de dependências do spring-boot-starter versão 2.1.3.RELEASE

Já imaginou ficar procurando e fazendo download de todas essas bibliotecas uma a uma (lembrando que há ainda várias não vistas na figura)?

Para situações como essa o **Apache Maven**, ou simplesmente **Maven**, ajuda imensamente. Ele é uma **build tool** que permite automatizar várias tarefas relativas à configuração e construção de um projeto Java. Suas possibilidades são muitas, mas nosso foco aqui é a funcionalidade de **gerenciamento de dependências** de bibliotecas.

O Maven é apenas uma interface de linha de comando (**CLI**), com o comando **mvn**. Porém, as principais IDEs Java abstraem bastante seu uso por meio de assistentes e atalhos. Para configurar o Maven num projeto, usa-se apenas um arquivo no formato XML, que é o **pom.xml**.

Existem alternativas ao Maven no ecossistema Java, embora ele ainda seja o líder com folga no mundo das *build tools*. Seu maior concorrente é o **Gradle**, que usa arquivos em Groovy ao invés de XML em sua configuração. Veja o nível de adoção das diferentes *build tools* Java segundo a pesquisa “*The State of Java in 2018*” na Figura 2.

Build Tool Adoption in 2018

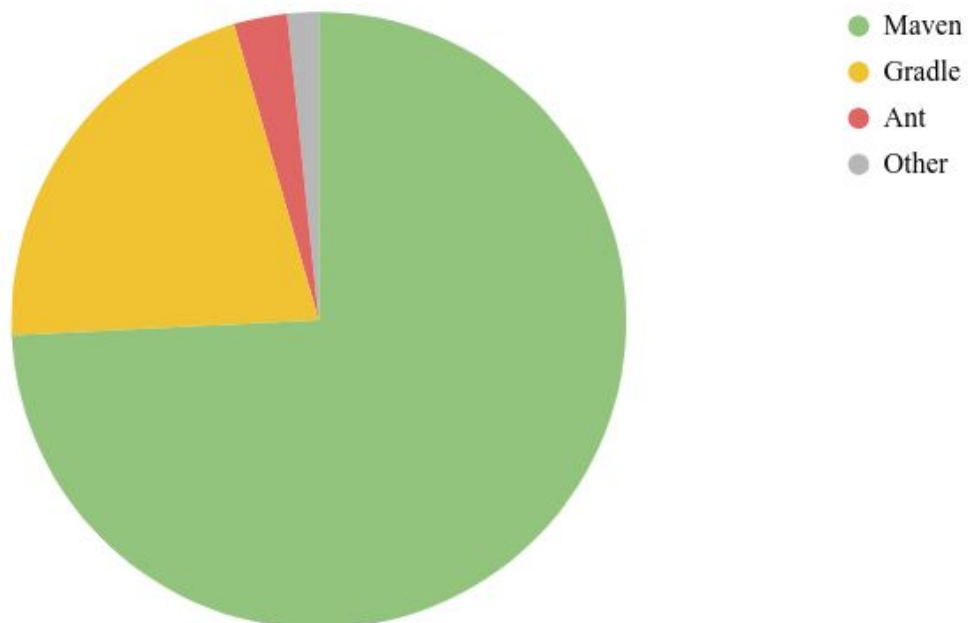


Figura 2: Adoção das build tools da plataforma Java em 2018

Fonte: <https://www.baeldung.com/java-in-2018>

Instalação do Maven

A seguir, veremos as maneiras de instalar e usar o Maven. Porém, para todas elas existe um pré-requisito em comum: possuir um **JDK** instalada.

Em IDEs

As principais IDEs Java (Eclipse, IntelliJIdea e NetBeans) já vêm com um Maven embarcado pronto para uso. Portanto, basta aprender como usar os comandos Maven nos seus assistentes.

No Windows

Caso você queira usar o comando **mvn** no prompt do Windows ou no PowerShell, você deve:

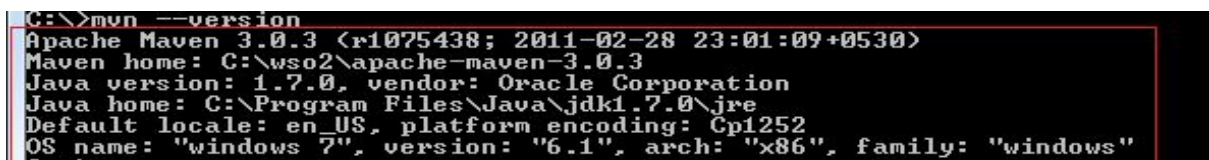
1. Fazer o download da versão do Maven que precisar em <http://maven.apache.org/download.cgi> (“*Binary **zip** archive*” ou “*Binary **tar.gz** archive*”)
2. Descompactar o arquivo onde preferir
3. Colocar o subdiretório **bin** de onde descompactou no **path** do Windows
4. Para testar, abra uma janela do prompt ou do PowerShell e execute o comando...

```
mvn --version
```

ou, o comando

```
mvn -v
```

Se tudo der certo, você verá uma saída como a da Figura 3.



```
C:\>mvn --version
Apache Maven 3.0.3 (r1075438; 2011-02-28 23:01:09+0530)
Maven home: C:\wso2\apache-maven-3.0.3
Java version: 1.7.0, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.7.0_jre
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 7", version: "6.1", arch: "x86", family: "windows"
```

Figura 3: Saída do comando “mvn --version” no prompt do Windows

No Linux

Caso você queira usar o comando **mvn** no terminal do Linux, você pode fazer a instalação manual ou via pacote.

Instalação manual:

1. Fazer o download da versão do Maven que precisar em <http://maven.apache.org/download.cgi> (“*Binary **zip** archive*” ou “*Binary **tar.gz** archive*”)
2. Descompactar o arquivo onde preferir
3. Colocar o subdiretório **bin** de onde descompactou no **PATH** do Linux
4. Conceda permissão de execução a todos os arquivos desse diretório **bin**
5. Para testar, abra um terminal e execute o comando...

```
mvn --version
```

ou, o comando

```
mvn -v
```

Se tudo der certo, você verá uma saída como a da Figura 4.

```
Apache Maven 3.5.0
Maven home: /usr/share/maven
Java version: 1.8.0_171, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-8-openjdk-amd64/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "4.13.0-46-generic", arch: "amd64", family: "unix"
```

Figura 4: Saída do comando “mvn -v” no terminal Linux

Instalação via pacote:

Use o gerenciador de pacotes de sua distribuição Linux (apt, yum etc) e instale o pacote do Maven. O nome varia por distribuição, podendo ser maven, mvn, dentre outros.

Feita a instalação, para testar, abra um terminal e execute o comando...

```
mvn --version
```

ou, o comando

```
mvn -v
```

Se tudo der certo, você verá uma saída como a da Figura 4.

No macOS

Caso você queira usar o comando **mvn** no terminal do macOS, você pode fazer a instalação manual ou via pacote.

Instalação manual:

1. Fazer o download da versão do Maven que precisar em <http://maven.apache.org/download.cgi> ("*Binary **zip** archive*" ou "*Binary **tar.gz** archive*")
2. Descompactar o arquivo onde preferir
3. Colocar o subdiretório **bin** de onde descompactou no **PATH** do macOS
4. Conceda permissão de execução a todos os arquivos desse diretório **bin**
5. Para testar, abra um terminal e execute o comando...

```
mvn --version
```

ou, o comando

```
mvn -v
```

Se tudo der certo, você verá uma saída como a da Figura 4.

Criação de um projeto Maven no NetBeans

Para criar um projeto Maven no NetBeans, devemos pedir a criação de um novo **Project**, depois, **Maven** -> **Java Application**, como na Figura 5.

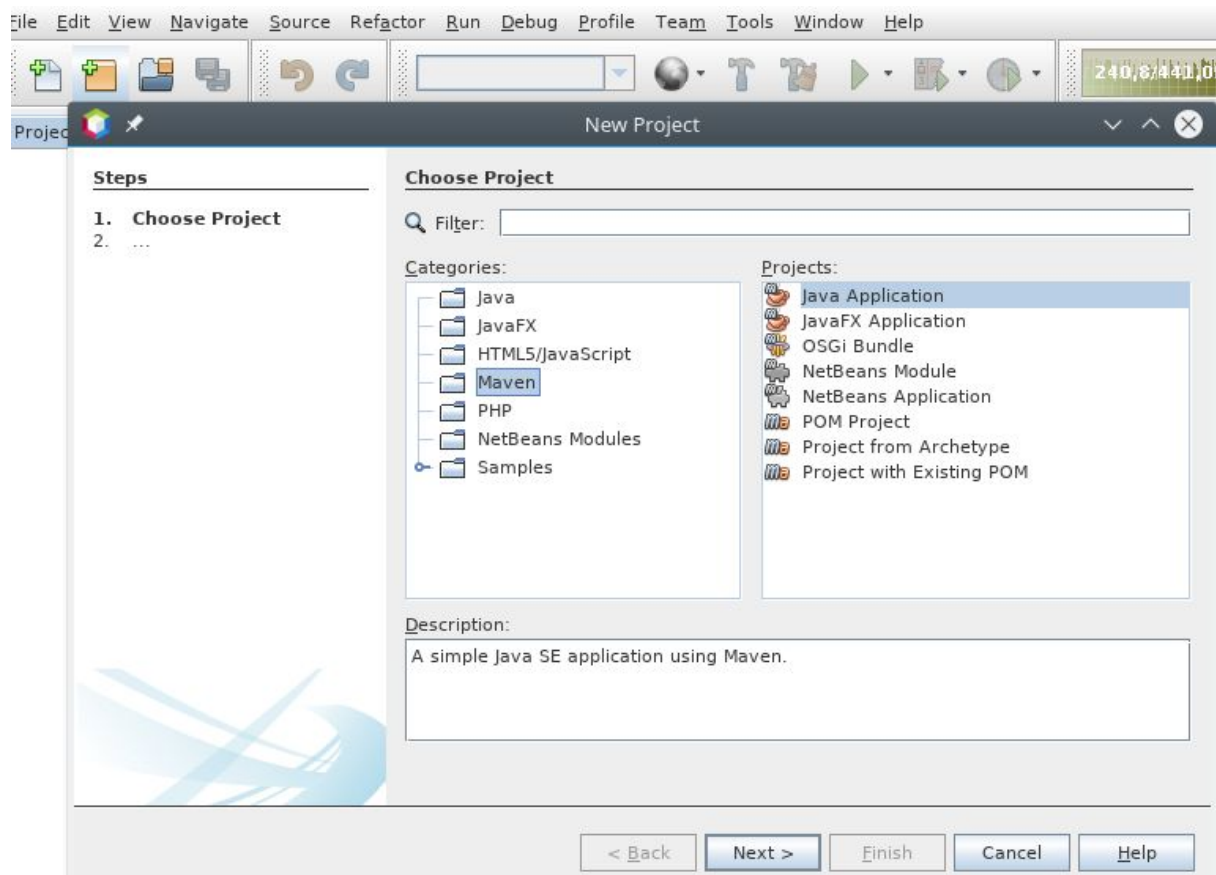


Figura 5: Assistente de criação de novo “Maven Project” no NetBeans

Após escolher as opções indicadas, uma janela com o assistente de criação de Maven Project irá surgir. Vide a Figura 6.

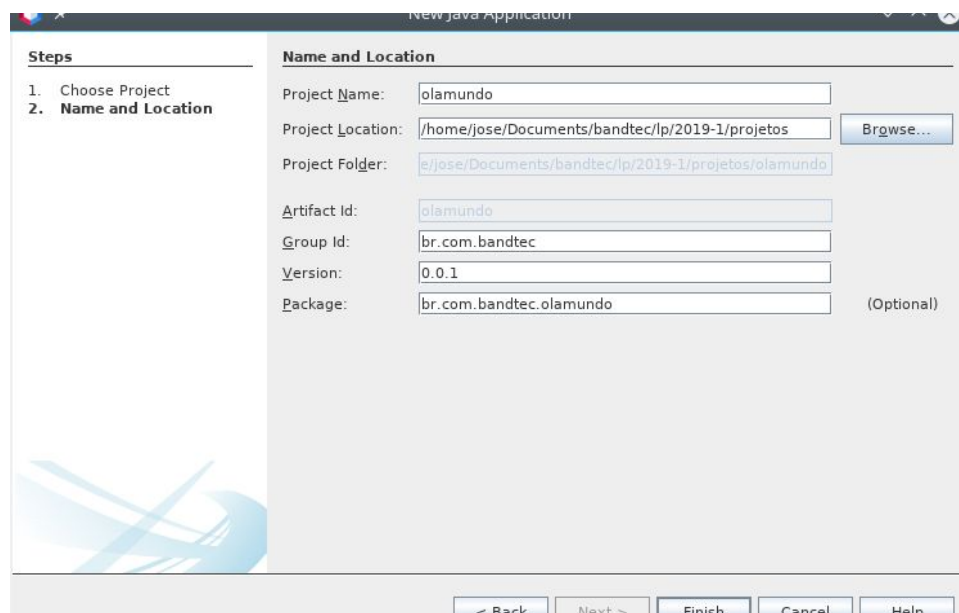


Figura 6: Primeira tela do assistente de criação de novo “Maven Project” no NetBeans

Nessa etapa, devemos informar os dados básicos de um projeto Maven, que são:

Group Id: Aqui é comum usar o “domínio” da empresa ou pessoa na internet (ex: **br.com.meudominio**). Outras pessoas usam apenas um nome de um grupo de projetos (exs: **projetos-matematicos** ou **minhaempresa**).

Artifact Id: É o nome do projeto em si. Para evitar projetos com o mesmo nome pelo mundo, o Maven considera que o “nome completo” é o **Group Id** junto ao **Artifact Id**.

Version: Versão do projeto. Aqui a recomendação é usar o padrão **Semantic Versioning**¹.

Package: O pacote Java que será criado no projeto (é comum que seja igual ao valor em **Group Id** seguido do valor em **Artifact Id**).

Com o assistente finalizado, o projeto criado terá uma estrutura na aba **Projects** como a da Figura 7.



Figura 7: Projeto Maven recém criado no NetBeans

Na aba **Projects** temos uma visão cheia de abstrações. Para ver como realmente está a estrutura de pastas e arquivos do projeto Maven recém criado, veja a Figura 8, que mostra a visão da aba **Files**.

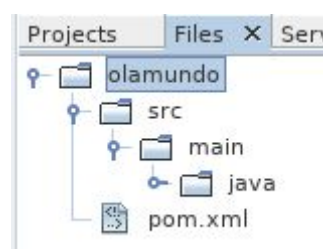


Figura 8: Estrutura real do Projeto Maven recém criado no NetBeans

¹ **Semantic Versioning** - <https://semver.org/>

A estrutura de diretórios e arquivos da Figura 8 representa o padrão de um projeto Maven. A seguir, os principais diretórios numa estrutura de um projeto Maven.

Diretório **src**: Código fonte do projeto.

Diretório **src/main/java**: Código fonte de arquivos na linguagem **Java**. Caso você use outras linguagens da JVM no projeto, pode existir, por exemplo um **src/main/groovy** (se usar a linguagem **Groovy** para algumas classes).

Diretório **src/main/resources**: Código fonte de arquivos não compiláveis (.properties, .xml, imagens, .pdf etc). No NetBeans, esse diretório não é criado pelo assistente.

Diretório **src/main/webapp**: Arquivos de front-end Web (apenas para projetos Web). No NetBeans, esse diretório não é criado pelo assistente.

Diretório **src/test/java**: Código fonte de arquivos de testes automatizados na linguagem **Java**. Caso você use outras linguagens da JVM no projeto, pode existir, por exemplo um **src/test/groovy** (se usar a linguagem **Groovy** para algumas classes de testes). No NetBeans, esse diretório não é criado pelo assistente.

Diretório **src/test/resources**: Código fonte de arquivos não compiláveis usados pelos testes automatizados (.properties, .xml, imagens, .pdf etc). No NetBeans, esse diretório não é criado pelo assistente.

Diretório **target**: Criado somente após a primeira construção do projeto. Para lá vão os arquivos compilados e/ou o arquivo resultante do empacotamento do projeto.

E, fora de qualquer diretório, ou seja, no diretório raiz do projeto, temos o arquivo **pom.xml**, sobre o qual falaremos melhor a seguir.

O pom.xml

O arquivo **pom.xml** é o **arquivo de configuração do Maven** do projeto. Nele pode ser configurado tudo que o podemos fazer com o Maven. No projeto de exemplo que criamos, o seu conteúdo seria como o do código a seguir.

```
<project ...>

<modelVersion>4.0.0</modelVersion>

<groupId>br.com.bandtec</groupId>

<artifactId>olamundo</artifactId>

<version>0.0.1</version>

<packaging>jar</packaging>

<properties>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<maven.compiler.source>1.8</maven.compiler.source>
<maven.compiler.target>1.8</maven.compiler.target>
</properties>

</project>
```

Esse **pom.xml** criado possui as principais e mais comuns *tags* de configurações usadas num projeto Maven. A seguir, a explicação para cada uma delas.

<project>: Tag Raiz de um pom.xml. Obrigatória.

<modelVersion>: Versão de modelo de configuração. Desde o Maven 2 usa-se 4.0.0 como valor. Obrigatória.

<groupId>: Valor do **Group Id** do projeto, conforme explicado no momento de sua criação. Obrigatória.

<artifactId>: Valor do **Artifact Id** do projeto, conforme explicado no momento de sua criação. Obrigatória.

<version>: Valor da **Versão** do projeto, conforme explicado no momento de sua criação. Obrigatória.

<packaging>: Tipo de arquivo no qual o projeto será “empacotado” quando construído. Opcional. Valor padrão: **jar**. Valores possíveis: **jar**, **war**, **pom**, **maven-plugin**, **ejb**, **ear** e **rar**.

<properties>: Tag que contém uma ou mais propriedades de configuração. Opcional.

Algumas tags importantes que não são criadas pelo assistente do NetBeans, mas que estariam logo após a **<properties>**.

<dependencies>: Tag com as dependências de outras bibliotecas Maven. Opcional (embora seja quase impossível ver projetos reais sem ela).

<dependency>: Tag com cada biblioteca que é dependência do projeto. É possível haver quantas tags dessa dentro de **<dependencies>** forem necessárias.

Aqui informa-se o conjunto **Group Id**, **Artifact Id** e **Version** da biblioteca em tags homônimas.

Também informa-se o escopo (tag **<scope>**), que indica de onde a biblioteca é obtida ou em qual momento da construção do projeto ela será “visível” (acessível).

Essa tag é opcional (embora seja quase impossível ver projetos reais sem ela).

Escopo de uma dependência

O **escopo** (tag **<scope>** dentro da tag **<dependency>**) indica, basicamente, de onde a biblioteca é obtida ou em qual momento da construção do projeto ela será “visível” (acessível). A seguir, os escopos de uma biblioteca num projeto Maven.

compile: É o **escopo padrão**, usado se nenhum for especificado. As classes dessa biblioteca e das suas dependências ficam disponíveis no projeto como se estivessem nele. É um jeito de dizer que determinada biblioteca será incorporada pelo projeto.

provided: Nesse escopo não é feito o download da biblioteca no momento da construção do projeto. Usamos esse escopo para bibliotecas fornecidas por

algum agente do ambiente onde o projeto será executado. Ex: Bibliotecas de um servidor de aplicação, como o Tomcat.

runtime: Muito parecido com o compile. A diferença é que as classes da biblioteca e suas dependências não serão usadas no momento da compilação do projeto.

test: Neste escopo, é feito o download da biblioteca, porém ela não é empacotada junto do projeto. Usamos esse escopo para bibliotecas usadas na criação de testes automatizados mas que não precisam ir para o pacote que será levado para produção.

system: É muito parecido com o compile. A diferença é que é usado para bibliotecas que não estão na internet e sim em algum lugar no computador onde o projeto será construído. Se usar ele, é necessário usar a tag **<systemPath>** dentro de **<dependency>** para indicar o caminho do arquivo .jar da biblioteca desejada.

Exemplos de dependência no pom.xml

Vamos supor que precisemos incluir uma biblioteca chamada **Apache Commons IO**. Trata-se de uma biblioteca que traz classes que facilitam imensamente tarefas de criação e escrita de arquivos em Java.

É possível ver as versões disponíveis no repositório online Maven em **<https://mvnrepository.com/artifact/commons-io/commons-io>**. Vamos usar a versão **2.6**. Assim, nosso **pom.xml** ficaria como o código de exemplo a seguir.

após o `</properties>`

```
<dependencies>
  <dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.6</version>
  </dependency>
</dependencies>
```

Agora vamos supor que precisemos incluir uma segunda dependência, a **ImageJ**, que facilita o tratamento de imagens com Java. É possível ver as versões disponíveis no repositório online Maven em <https://mvnrepository.com/artifact/net.imagej/ij>. Vamos usar a versão **1.52n**. Assim, nosso **pom.xml** ficaria como o código de exemplo a seguir.

```
após o </properties>

<dependencies>
    <dependency>
        <groupId>commons-io</groupId>
        <artifactId>commons-io</artifactId>
        <version>2.6</version>
    </dependency>

    <dependency>
        <groupId>net.imagej</groupId>
        <artifactId>ij</artifactId>
        <version>1.52n</version>
    </dependency>
</dependencies>
```

Agora vamos supor que precisemos incluir uma terceira dependência, a **JUnit**, que facilita a criação de testes automatizados de código. É possível ver as versões disponíveis no repositório online Maven em <https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-api>. Vamos usar a versão **5.4.1**.

Detalhe importante: Essa dependência ficará no escopo **test**. Ou seja, essa biblioteca não ficará junto do “pacote” do projeto (costumamos dizer que essa biblioteca “não vai para o ambiente de produção”. O Maven só usará essa biblioteca quando executar os testes automatizados.

Assim, nosso **pom.xml** ficaria como o código de exemplo a seguir.

```
após o </properties>

<dependencies>
  <dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.6</version>
  </dependency>

  <dependency>
    <groupId>net.imagej</groupId>
    <artifactId>ij</artifactId>
    <version>1.52n</version>
  </dependency>

  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.4.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

De onde vêm as bibliotecas nas <dependency>

Você deve ter notado que uma dependência configurada em **<dependency>** é também um projeto Maven. Assim que precisar da biblioteca, o Maven procura ela num diretório chamado **.m2** que fica no diretório do usuário do sistema operacional. A esse diretório chamamos **repositório local Maven**. Caso não encontre lá, faz o download de algum **repositório online de bibliotecas Maven**.

O repositório online de bibliotecas Maven padrão é o **Maven Central**, que fica em **<http://central.maven.org/maven2/>**. É possível configurar outros repositórios, caso necessário, mas foge ao escopo de nossos estudos por agora.

Uma vez que uma determinada versão de biblioteca é usada por um usuário de SO, ela fica no **.m2**. Assim, caso seja criado outro projeto com a mesma dependência, não será feito novamente o download.

A vantagem dessa abordagem é que as bibliotecas (arquivos .jar) não ficam fisicamente junto do projeto, o que poupa muito espaço em repositórios de controle de versão como o **git**, por exemplo.

Como descubro as informações de bibliotecas que preciso usar

Agora que já sabe que as bibliotecas das dependências vêm quase sempre da internet, como sabe qual o **Group Id**, **Artifact Id** de uma biblioteca que preciso? E como saber quais versões são disponíveis?

É importante saber que o Maven é uma ferramenta consolidada da plataforma Java. Tão consolidada que é extremamente difícil que uma biblioteca que você precise não esteja no *Maven Central*, bastando você indicar seu grupo, artefato e versão nas dependências de seu projeto. Assim, basicamente basta saber o nome da biblioteca que precisa e pesquisar em sites que catalogam bibliotecas Maven, como o **MVN Repository** (<https://mvnrepository.com/>), por exemplo.

Construindo um projeto Maven

É possível construir, ou seja, compilar e gerar seu “pacote” usando assistentes da IDE ou via linha de comando. Independente do método usado, tudo que for gerado por esse processo vai para o diretório **target** do projeto.

Construindo via linha de comando

Para construir um projeto com a linha de comando, basta executar, **no diretório raiz do projeto**, o seguinte comando:

```
mvn install
```

Após a execução desse comando, a saída termina como o exemplo da Figura 9.

```
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 12.209 s
[INFO] Finished at: 2019-03-18T22:03:47-03:00
[INFO] Final Memory: 16M/158M
[INFO] -----
```

Figura 9: Final da saída do comando “mvn install”

Construindo com NetBeans

Caso você tenha acabado de incluir uma nova dependência no pom.xml, é necessário usar o assistente **Build with Dependencies**, como mostra a Figura 10. Esse assistente efetivamente faz o download (caso necessário) e “incorpora” a(s) nova(s) biblioteca(s) ao seu projeto.

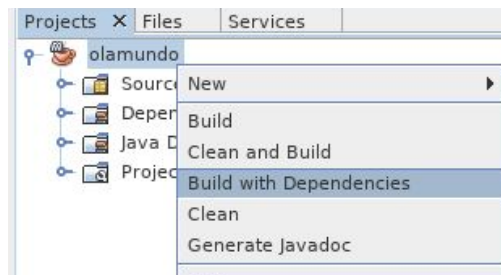


Figura 10: Solicitando um “Build with Dependencies” no NetBeans

Se algo deu errado na execução, verifique se informou corretamente os **groupId**, **artifactId** e **version** no **pom.xml**. Verifique ainda sua conexão com a internet, caso seja uma dependência que não estava em seu repositório local Maven.

Para construir um projeto no NetBeans, basta solicitar um tipo de execução chamado **Clean and Build** (ou apenas a **Build**, menos recomendável), como mostra a Figura 11.

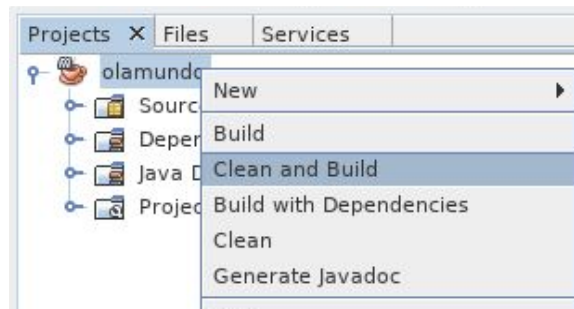


Figura 11: Solicitando a construção de um projeto Maven no NetBeans

Após a execução, a saída termina é exibida na janela “*Output*” do NetBeans como o exemplo da Figura 12.

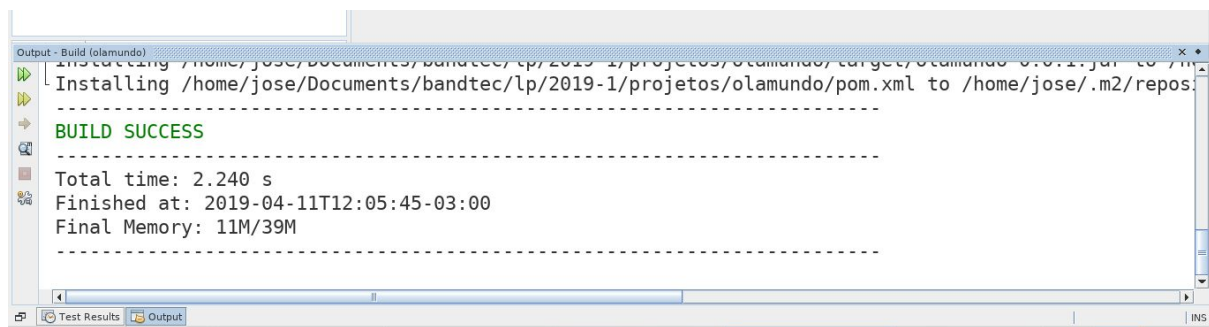


Figura 12: Final da saída da execução “Clean and Build” no NetBeans

Independentemente se sua execução foi via linha de comando ou usando a IDE, caso seja a primeira vez que fizer isso em um projeto ou logo após acrescentar uma dependência que nunca usou no computador, vai notar que será realizado o download de todas as dependências necessárias. Veja uma amostra de como o Maven descreve esses downloads na Figura 13.



Figura 13: Log de download de dependências logo após um comando “mvn install”