



(<http://luizricardo.org/>)

## **Instalando, configurando e usando o Maven para gerenciar suas dependências e seus projetos Java** **(<http://luizricardo.org/2014/06/instalando-configurando-e-usando-o-maven-para-gerenciar-suas-dependencias-e-seus-projetos-java/>)**

---

Este artigo é uma introdução ao Maven (<http://maven.apache.org/>), uma ferramenta de gerenciamento de dependências e ciclo de vida de um projeto em Java.

Para usar o Maven, você precisa ter o JDK instalado. Você também pode integrá-lo à sua IDE. Para saber como instalar o JDK e o Eclipse, consulte os seguintes artigos:

- Instalando e Configurando o Java Development Kit 7 para Desenvolvimento (<http://luizricardo.org/2013/11/instalando-e-configurando-o-java-development-kit-7-para-desenvolvimento/>)
- Instalando, Configurando e Usando o Eclipse Kepler (<http://luizricardo.org/2013/11/instalando-configurando-e-usando-o-eclipse-kepler/>)

# O que é o Maven?

O Maven é uma ferramenta de gerenciamento de dependências e do ciclo de vida de projetos de software no sentido técnico. Isso inclui:

- 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.

## Conceitos importantes

### 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, como `br.com` (`http://br.com`). `starcode`, `org.apache`, `com` (`http://com`). `google`, `com` (`http://com`). `ibm`, etc.
- **Identificador único de artefato:** geralmente é o nome do projeto. Ele deve ser único por grupo.
- **Número de versão:** a versão do projeto, como `1.4.2` ou `3.0`. 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`. Se há variações específicas para Sistemas Operacionais, podemos ter os classificadores `linux` e `windows`.

## 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. Continue lendo para entender o que é isso.

O repositório possui uma estrutura padrão onde o Maven consegue encontrar os artefatos através da identificação do mesmo.

## 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 ([http://www.jfrog.com/home/v\\_artifactory\\_opensource\\_overview](http://www.jfrog.com/home/v_artifactory_opensource_overview)) ou Nexus (<http://www.sonatype.org/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.

## 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”.

## Ciclo de vida padrão do Maven

O Maven possui um ciclo de vida padrão. Cada passo do ciclo de vida é chamado de *Goal* e possui plugins específicos. Os mais importantes são:

- **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.

Os itens acima, nessa ordem, são passos comuns para geração de uma versão de qualquer sistema, não é mesmo?

No Maven, você pode configurar detalhadamente cada um desses passos e até incluir novos. Por exemplo, alguns frameworks que geram código-fonte usam um *goal* `generate-sources` para fazer isso.

Além disso, não é necessário executar todos os passos sempre. Você pode escolher qual deseja executar num determinado momento, mas o Maven sempre executará todos os passos anteriores.

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.

Para maiores informações sobre o ciclo de vida, consulte a documentação (<http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html>).

## 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.

Veja a seguir os principais diretórios utilizados:

- `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.

Para ver mais detalhes sobre a estrutura de diretórios do Maven, consulte a documentação (<http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html>).

## Usando o Maven em projetos já existentes

Você pode ter ficado desapontado com a estrutura anterior, pois estava pensando em usar o Maven em um projeto que já começou, mas não quer ou não pode mudar a estrutura de pastas atuais.

**Saiba que o Maven é flexível e permite alterar toda a estrutura padrão.**

Por exemplo, é comum usar a pasta `src` para os fontes, ao invés de `src/main/java`. Para ajustar isso, basta adicionar uma tag `<sourceDirectory>` dentro da tag `<build>`, assim:

```
<project>
  ...
  <build>
    <sourceDirectory>src</sourceDirectory>
    ...
  </build>
  ...
</project>
```

Não vou ir fundo nessa questão, mas se o leitor tiver um projeto em condições semelhantes, sugiro uma leitura mais completa da documentação, começando com *Using Maven When You Can't Use the Conventions* (<http://maven.apache.org/guides/mini/guide-using-one-source-directory.html>).

É claro que nem tudo é tão simples. Muitos projetos usam estruturas tão diferentes que se exige a refatoração desta estrutura.

## Benefícios do Maven

A adoção do Maven no desenvolvimento traz de imediato os seguintes benefícios:

### Centralização das informações

O Maven centraliza as informações dos projetos no arquivo pom.

Sendo assim, não é preciso configurar várias ferramentas, *build scripts*, servidores e IDEs durante o desenvolvimento. O Maven segue o conceito **DRY** (*Don't Repeat Yourself*).

Além disso, o Maven também disponibiliza formas de analisar o projeto. Por exemplo, o *goal* `dependency:analyze` exibe as dependências declaradas que não estão sendo usadas e as usadas via terceiros, mas não declaradas no pom.

## Padronização do ambiente de desenvolvimento

Através da especificação do projeto, incluindo suas características e dependências, o Maven constrói a estrutura necessária do projeto, baixando automaticamente as versões corretas das dependências (JARs, por exemplo) de um repositório central ou de um repositório privado (contendo sistemas da empresa).

Você não precisa entrar no site de cada biblioteca e framework usado e então fazer manualmente o download e adicionar os jars no seu *classpath*.

Dessa forma, cada desenvolvedor consegue configurar rapidamente um ambiente para desenvolvimento com a garantia de que esse ambiente é igual ao dos outros desenvolvedores.

## Gerenciamento de dependências

Como já mencionei, o Maven faz o download automático de dependências para o projeto e os adiciona ao *classpath* do seu projeto.

Cada dependência pode ter também as suas próprias dependências. Elas são chamadas **dependências transitivas**. O Maven resolve essa árvore de dependências e traz tudo o que você precisa.

Em alguns casos, podem haver problemas de conflitos, no caso da árvore de dependências incluir versões diferentes de um mesmo artefato. O Maven vem com mecanismos para resolver isso.

## Facilidade de compreensão do projeto

Ao usar a convenção de diretórios sugerida pelo Maven os desenvolvedores terão mais facilidade em compreender a estrutura do projeto, afinal todos os projetos seguirão uma estrutura básica de diretórios, como vimos anteriormente.

## Automação

O Maven gerencia o ciclo de vida da aplicação. Após configurar um projeto, você será capaz de executar comandos que vão desde a compilação até a geração de documentação (Javadoc) e um site padrão contendo informações sobre o projeto.

Uma vez feita a configuração necessária, o projeto pode ser baixado e compilado sem nenhum esforço. Novas versões podem ser geradas em servidores de Integração Contínua e testadas automaticamente sempre que necessário.

## **Um alerta**

Embora os tópicos anteriores tenham enumerado diversas vantagens do uso do Maven, este não é uma “bala de prata”, ou seja, uma solução mágica para o projeto.

Dependendo da complexidade do projeto, pode ser bem complicado criar uma configuração adequada para ao Maven.

Além disso, o Maven não irá livrá-lo de problemas como:

## **Incompatibilidade de dependências**

O projeto depende dos frameworks A e B. O framework A depende a versão 1.0 da biblioteca C. O framework B depende da versão 2.0 da biblioteca C.

O Maven não vai resolver sozinho isso, mas facilita muito a resolução do problema já que podemos usar as informações do mecanismo de resolução de dependências para identificar os conflitos.

## **Algumas tecnologias simplesmente não vão funcionar bem com o Maven**

Alguns autores de frameworks ou arquiteturas inventam o seu próprio jeito de trabalhar. Isso significa que para usar o Maven é necessário alguma adaptação, o que nem sempre é trivial.

No entanto, é possível escrever plugins para o Maven que façam o trabalho para você. Geralmente a comunidade de cada framework, se não os próprios autores, já terão resolvido esse problema. Embora existam casos em que essas soluções acrescentem novas limitações.

## **Ódio do Maven**

A verdade é que existe muita gente que odeia o Maven por ter vivido experiências ruins com ele, principalmente no início. Infelizmente, não sei se este artigo terá o poder de curá-lo de traumas passados ao tentar usar o Maven sem a devida orientação. 🙄

No entanto, não deixe que isso influencie você neste momento. Mesmo que não pretenda usar o Maven em seus projetos, vale a pena conhecê-lo. Você pode ser obrigado a usá-lo na empresa ou mesmo num projeto *opensource* de que vai participar.


Se você não gosta do Maven, tenha duas coisas em mente:

1. Existem várias alternativas, desde scripts Ant até outras ferramentas de resolução de dependências mais avançadas como o Graddle (<http://www.gradle.org/>).
2. Embora algumas pessoas atinjam um excelente nível de produtividade sem o Maven, se considerarmos um contexto mais amplo, como um projeto que com mais de meia dúzia de desenvolvedores, alguns deles novatos, o Maven pode trazer mais vantagens que desvantagens se bem configurado por um desenvolvedor experiente.

## Instalando o Maven


Acesse a página do Maven (<http://maven.apache.org/>) e clique no item Download (<http://maven.apache.org/download.cgi>) do menu.

A página disponibiliza diferentes versões para diferentes ambientes. Baixe o arquivo da última versão de acordo com seu sistema operacional. Destaquei na imagem a versão zip para Windows que usarei neste exemplo:



([http://luizricardo.org/wordpress/wp-content/uploads-files/2014/06/011.png](http://luizricardo.org/wordpress/wp-content/uploads/files/2014/06/011.png))

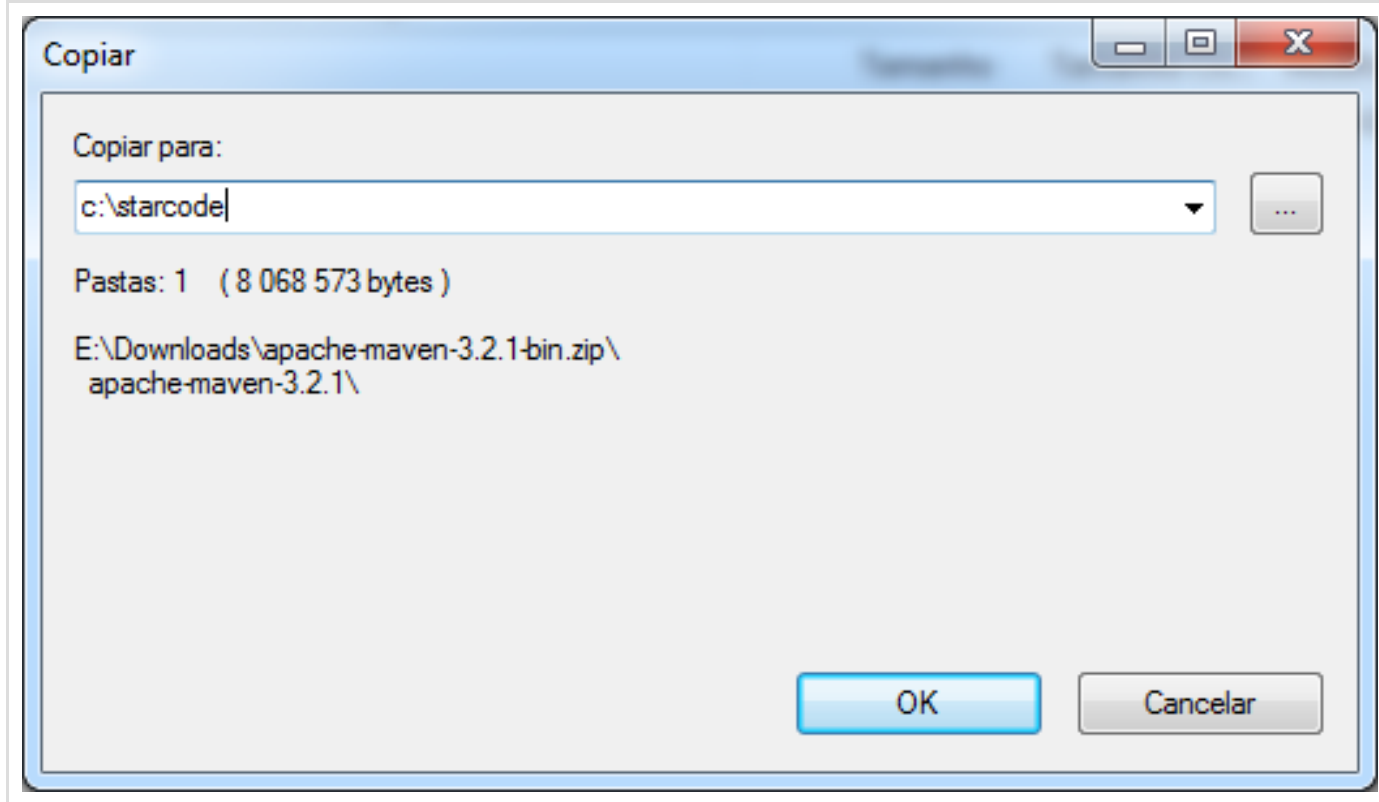
A versão mais atual do Maven na data de criação deste tutorial é 3.2.1. O pacote baixado é nomeado `apache-maven-3.2.1-bin.zip`. Veja o arquivo aberto no 7-Zip:



([http://luizricardo.org/wordpress/wp-content/uploads-files/2014/06/021.png](http://luizricardo.org/wordpress/wp-content/uploads/files/2014/06/021.png))

Descompacte o conteúdo para a pasta `c:\starcode\`.





(<http://luizricardo.org/wordpress/wp-content/uploads/files/2014/06/031.png>)

Configura o resultado na imagem a seguir:

(<http://luizricardo.org/wordpress/wp-content/uploads/files/2014/06/041.png>)

## Configurando o Maven

O Maven é configurado através do arquivo `settings.xml` que fica no diretório `c:\starcode\apache-maven-3.2.1\conf`.

Abra o arquivo usando um editor avançado, como o Notepad++ (<http://notepad-plus-plus.org/>). Você vai ver que existem diversos blocos de XML comentados com possíveis configurações e explicações sobre elas.

Em um ambiente simples você não vai precisar mexer em muita coisa. Porém, vamos ver alguns pontos mais importantes.

## Proxy

É muito comum precisarmos autenticar o acesso à internet em um Proxy quando estamos no trabalho. Procure a tag `<proxy>`, a qual deve estar comentada no arquivo de configuração. O trecho é o seguinte:

```
<proxy>
  <id>optional</id>
  <active>true</active>
  <protocol>http</protocol>
  <username>proxyuser</username>
  <password>proxypass</password>
  <host>proxy.host.net</host>
  <port>80</port>
  <nonProxyHosts>local.net|some.host.com</nonProxyHosts>
</proxy>
```

Se você tem um proxy na sua rede, mova o bloco acima para fora do comentário, então substitua os parâmetros de acordo com seu ambiente. Mantenha a tag `<proxy>` dentro de `<proxies>`.

Veja abaixo um exemplo de uso:

```
<proxies>
  <proxy>
    <id>proxy</id>
    <active>true</active>
    <protocol>http</protocol>
    <host>proxy.intranet.empresa.com</host>
    <port>8080</port>
    <nonProxyHosts>localhost,127.*,192.*</nonProxyHosts>
  </proxy>
</proxies>
```

## Local do repositório

O Maven utiliza um diretório local para baixar os artefatos da internet. O diretório padrão fica dentro pasta do usuário, na pasta `.m2`. Um exemplo no Windows é `c:\users\luiz\.m2\repository`.

Entretanto, tenho o hábito de mudar esse diretório para junto de meus arquivos de desenvolvimento. Para isso, basta editar o `settings.xml`, movendo a tag `<localRepository>` para fora do comentário e adicionando o caminho, por exemplo:

```
<localRepository>c:\starcode\apache-maven-3.2.1\repo</localRe
pository>
```

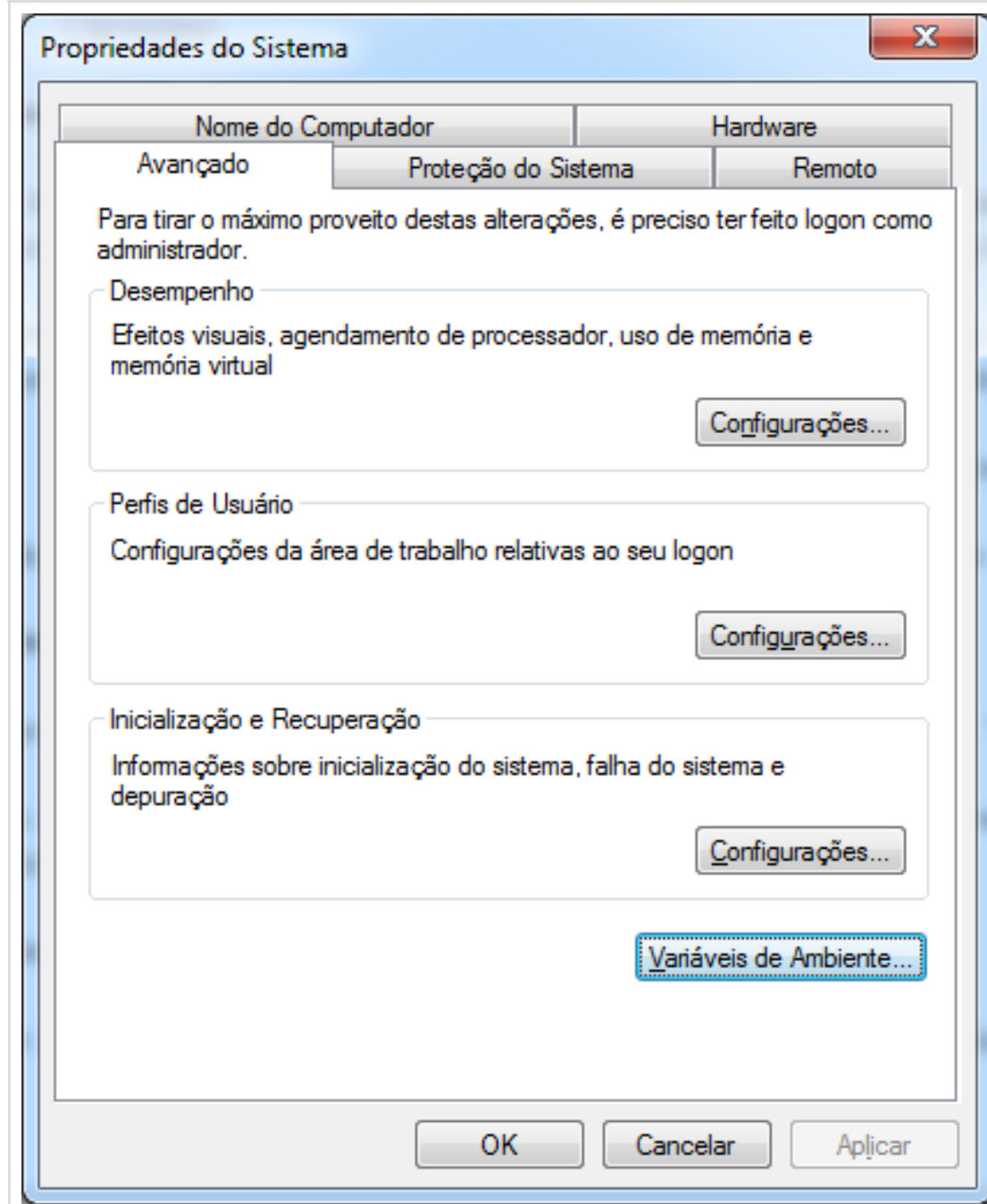
Não se esqueça de criar o diretório especificado caso o mesmo não exista.

## Configurando as variáveis de ambiente

Para usar o Maven em linha de comando você deve adicionar o caminho para os executáveis ao `PATH` do ambiente. No Windows, pressione `Ctrl+Break` para abrir a tela de informações do sistema.

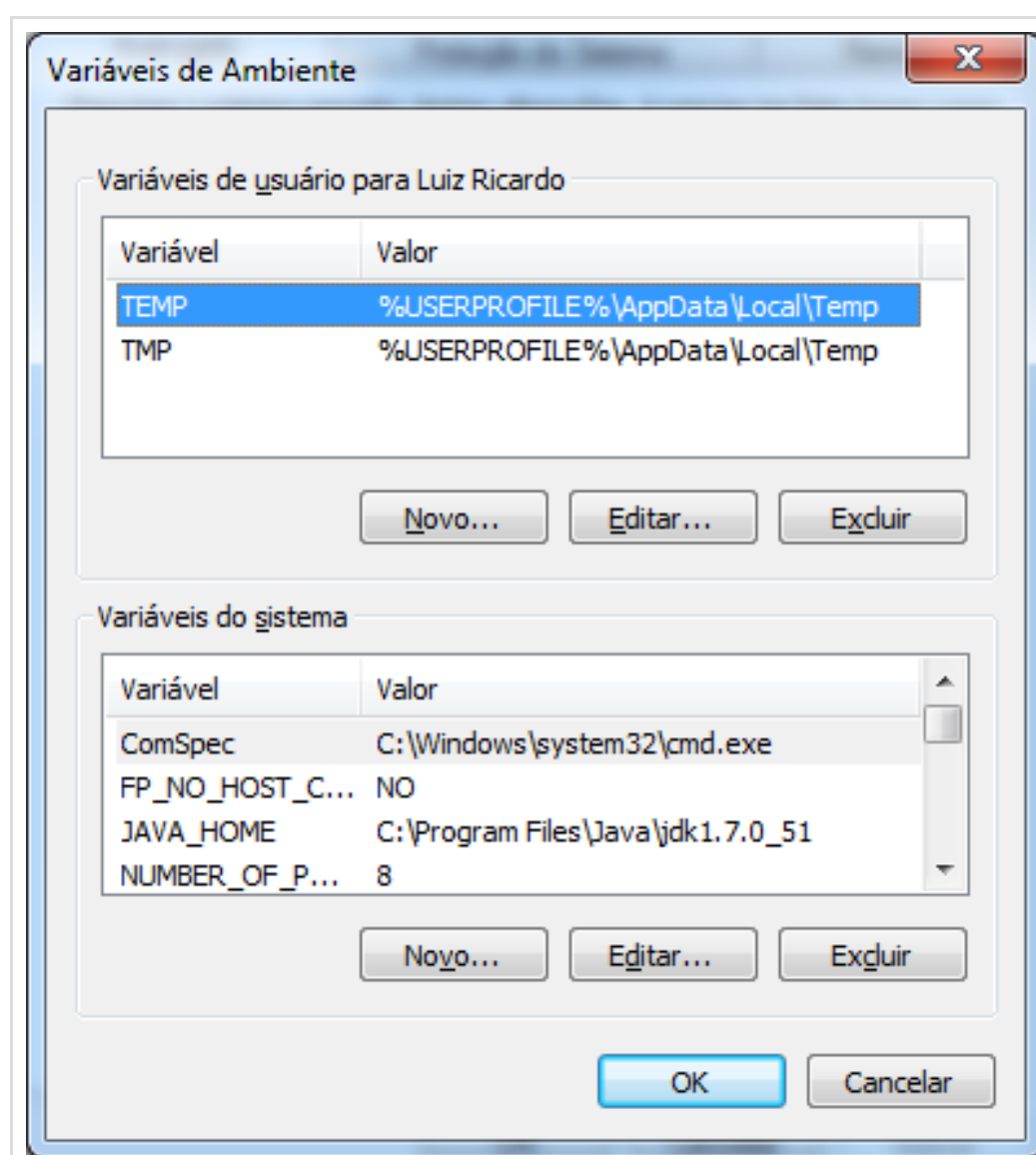
(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/05-configurar-maven.png>)

Clique na opção `Configurações avançadas do sistema`, à esquerda da janela.



(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/06-configurar-maven.png>)

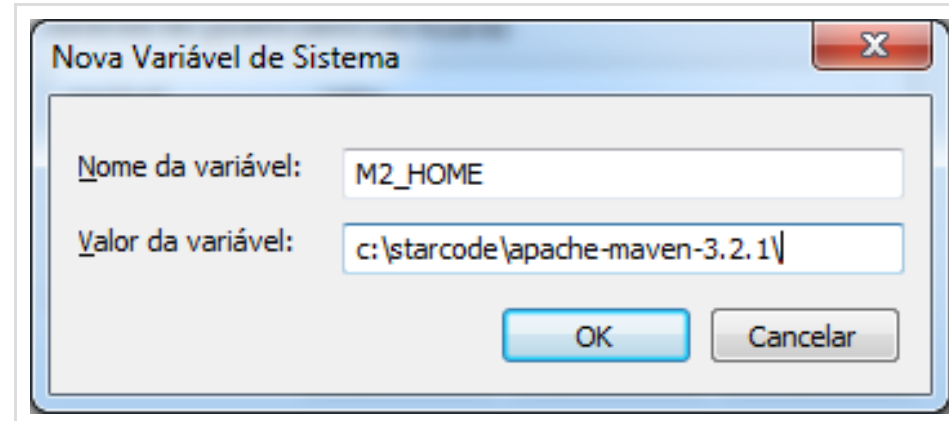
Na aba Avançado da tela de **Propriedades do Sistema**, clique em Variáveis de Ambiente....



(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/07-configurar-maven.png>)

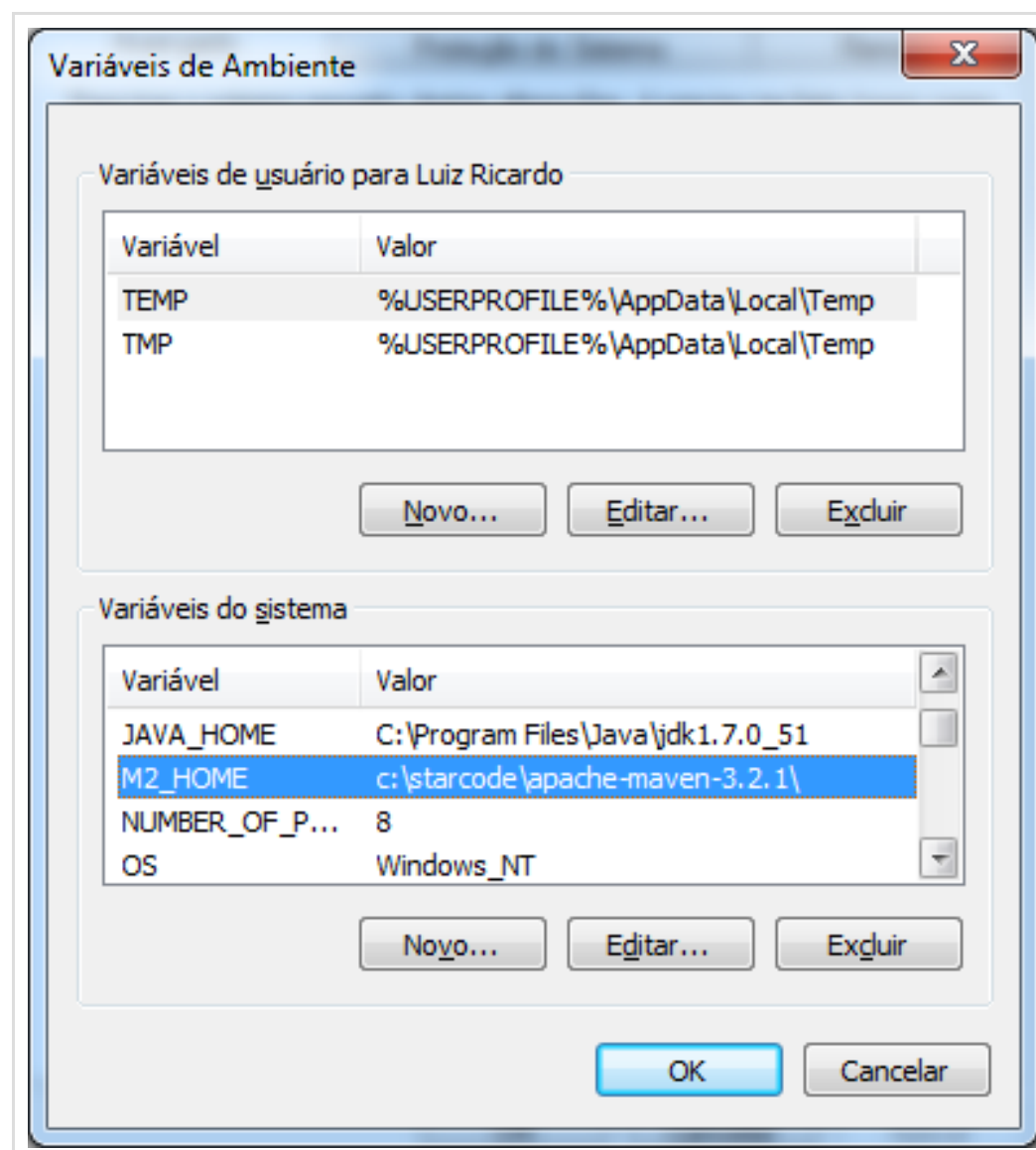
Você pode adicionar a variável de ambiente apenas para o usuário atual ou globalmente para todo o sistema. Faça o que for melhor para o seu caso. Alguns ambientes corporativos impedem o acesso à configuração de sistema por questões de segurança, então você terá que configurar apenas seu usuário.

Clique em **Novo...** e crie a variável `M2_HOME` com o valor apontando para o diretório base do Maven. No nosso exemplo o valor é `c:\starcode\apache-maven-3.2.1\`.



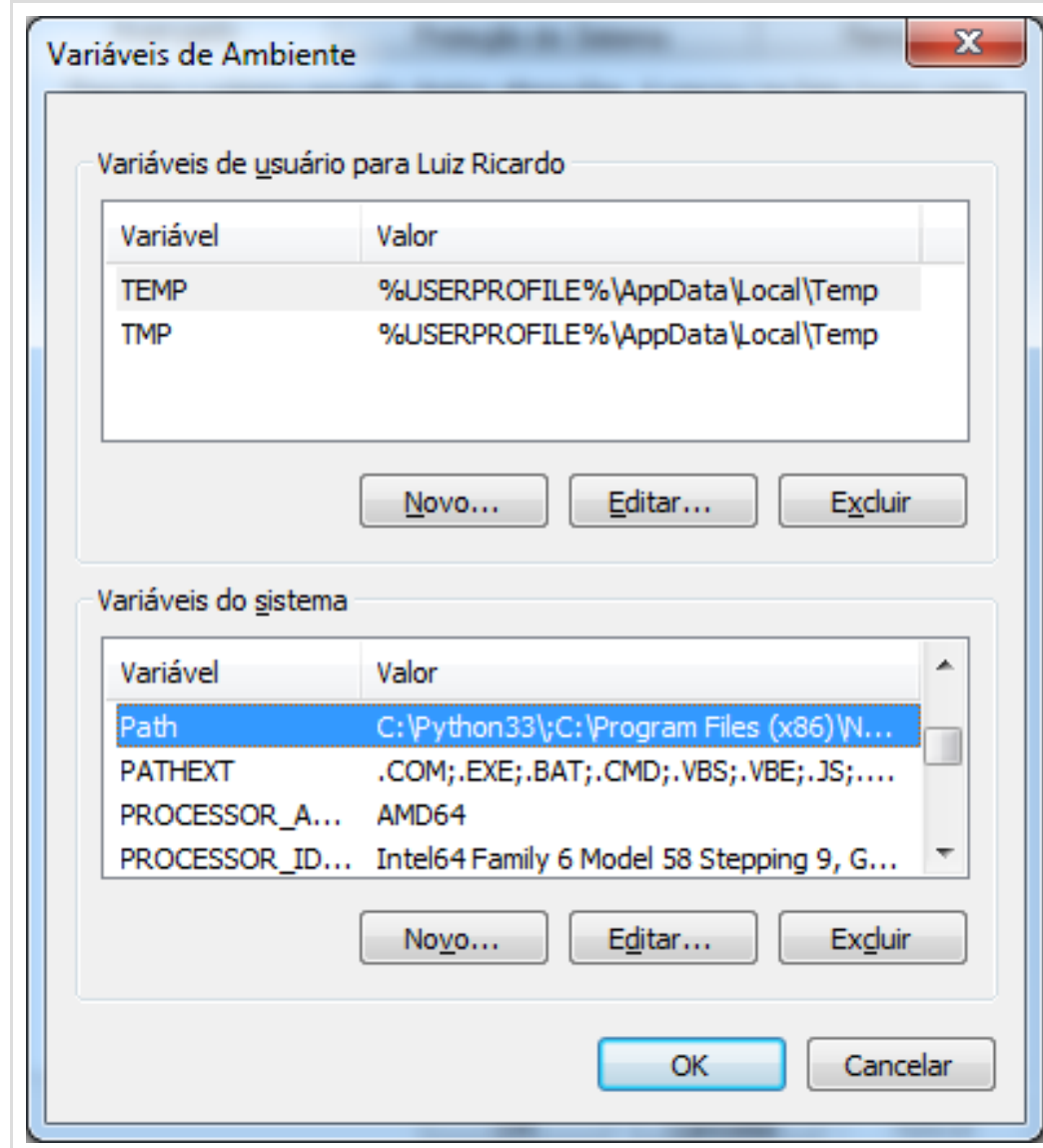
(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/08-configurar-maven.png>)

Clique em **OK** para criar a variável.



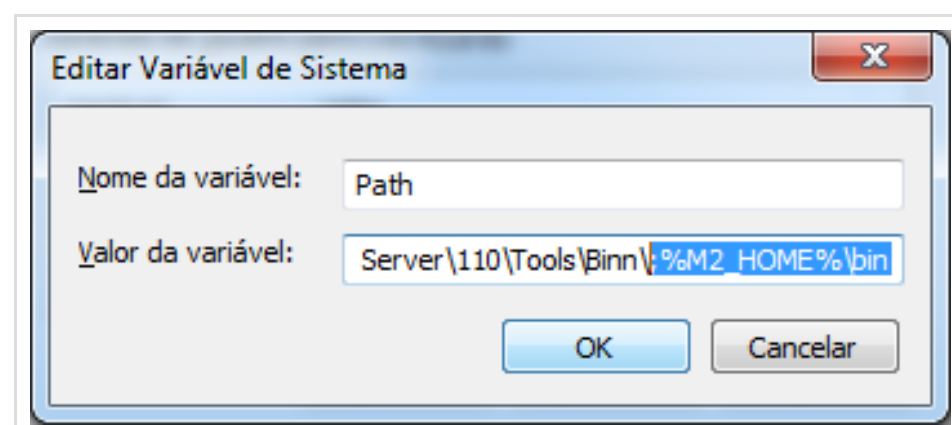
(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/09-configurar-maven.png>)

Agora vamos incluir o diretório com os executáveis do Maven ao `PATH`. Localize a entrada, selecione-a e clique em **Editar...**.



(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/10-configurar-maven.png>)

Adicione ao final um ponto e vírgula e o caminho para a pasta bin do Maven ( ;%M2\_HOME%\bin ), assim:



(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/11-configurar-maven.png>)

Clique em OK para confirmar a edição e OK novamente para confirmar as alterações nas variáveis do sistema.

Vamos então testar a instalação. Abra o **CMD** (linha de comando) e digite `mvn -version`. Você deve ver algo como na figura abaixo:

(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/12-configurar-maven.png>)

Se ocorreu algum problema, verifique se você tem o Java instalado e configurado corretamente, incluindo as variáveis de ambiente `JAVA_HOME` e `PATH` incluindo o Java. Caso tenha dúvidas, acesse o artigo citado no início sobre a instalação do JDK.

## Usando o Maven

# Usando a instalação do Maven no Eclipse

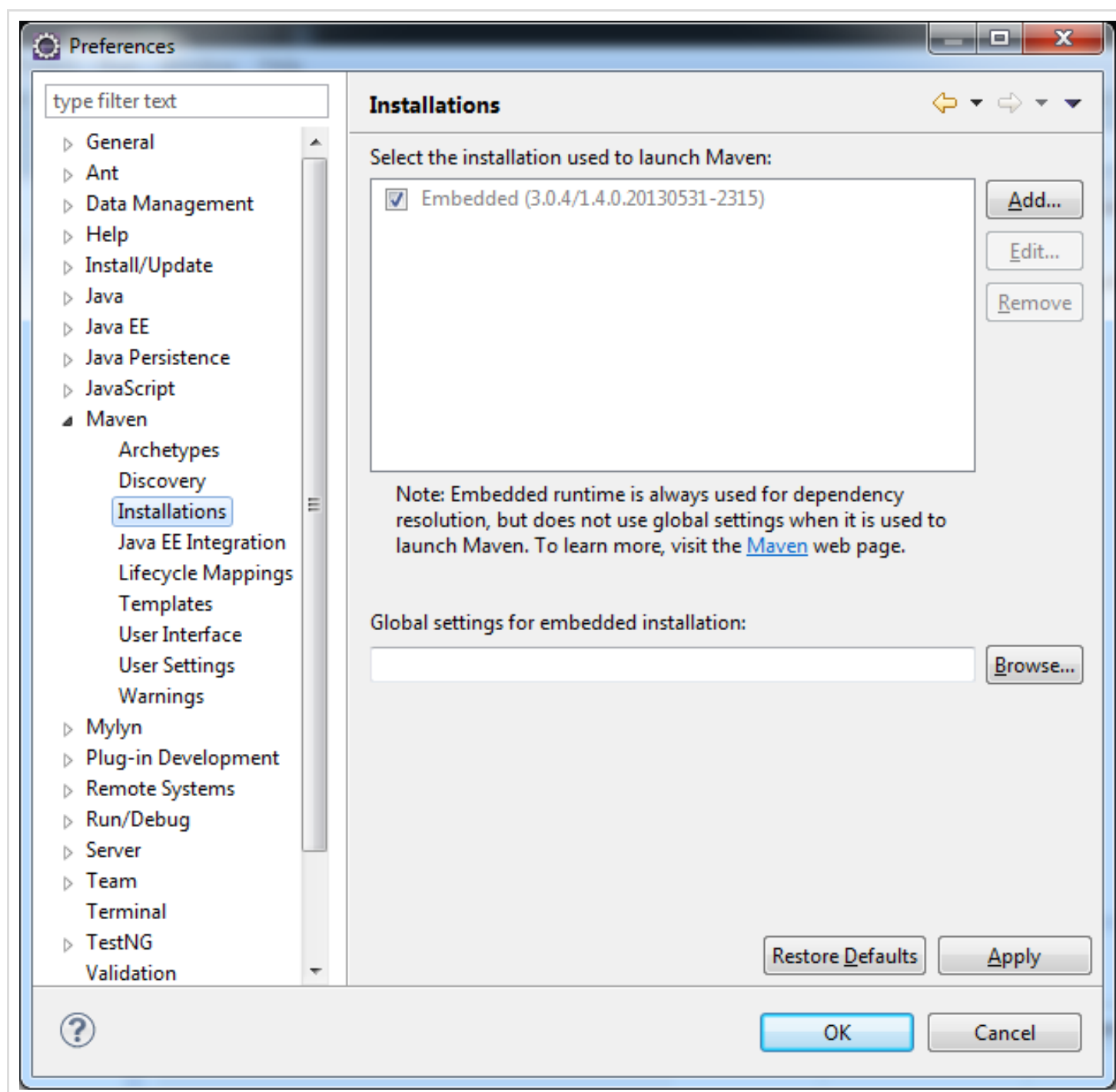
Para integrar o Maven e Eclipse eu aconselho o plugin M2E. Note que o M2E é um plugin do Eclipse que faz integração com o Maven.

Existe também o **Maven Eclipse Plugin**, aquele onde se digita `eclipse:eclipse` para gerar um projeto para o Eclipse. Este é um **plugin do Maven** que simplesmente gera os arquivos de configuração necessários para a IDE. Não confunda os dois.

A distribuição Eclipse for JEE Developers já vem com o plugin M2E e uma instalação interna do Maven. Veja como instalar e usar o Eclipse acessando o artigo citado no início.

Se você tem uma versão diferente do Eclipse, use o menu `Help > Eclipse Marketplace...`, pesquise por M2E e instale o plugin.

Com o plugin instalado e o Eclipse aberto, acesse o menu `Window > Preferences...` e vá até a opção `Maven > Installations`.

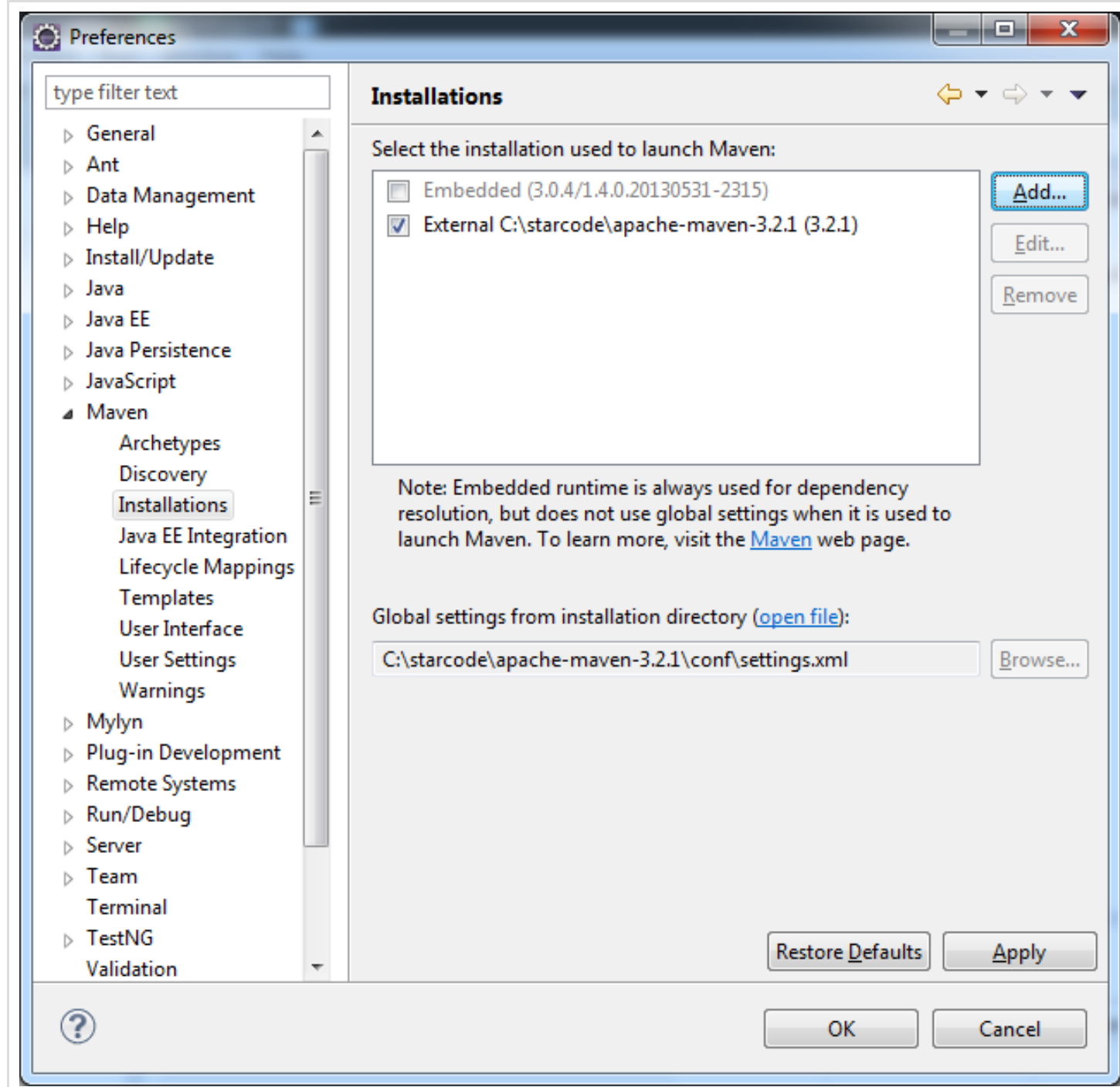


(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/01-maven-eclipse.png>)

Veja que já existe uma instalação “embarcada”, mas com uma versão desatualizada. Vamos adicionar o nosso Maven.

Clique em `Add...` e selecione a pasta com a nossa instalação, no caso: `c:\starcode\apache-maven-3.2.1`.





(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/02-maven-eclipse.png>)

Note que ele já encontrou nosso arquivo de configuração.

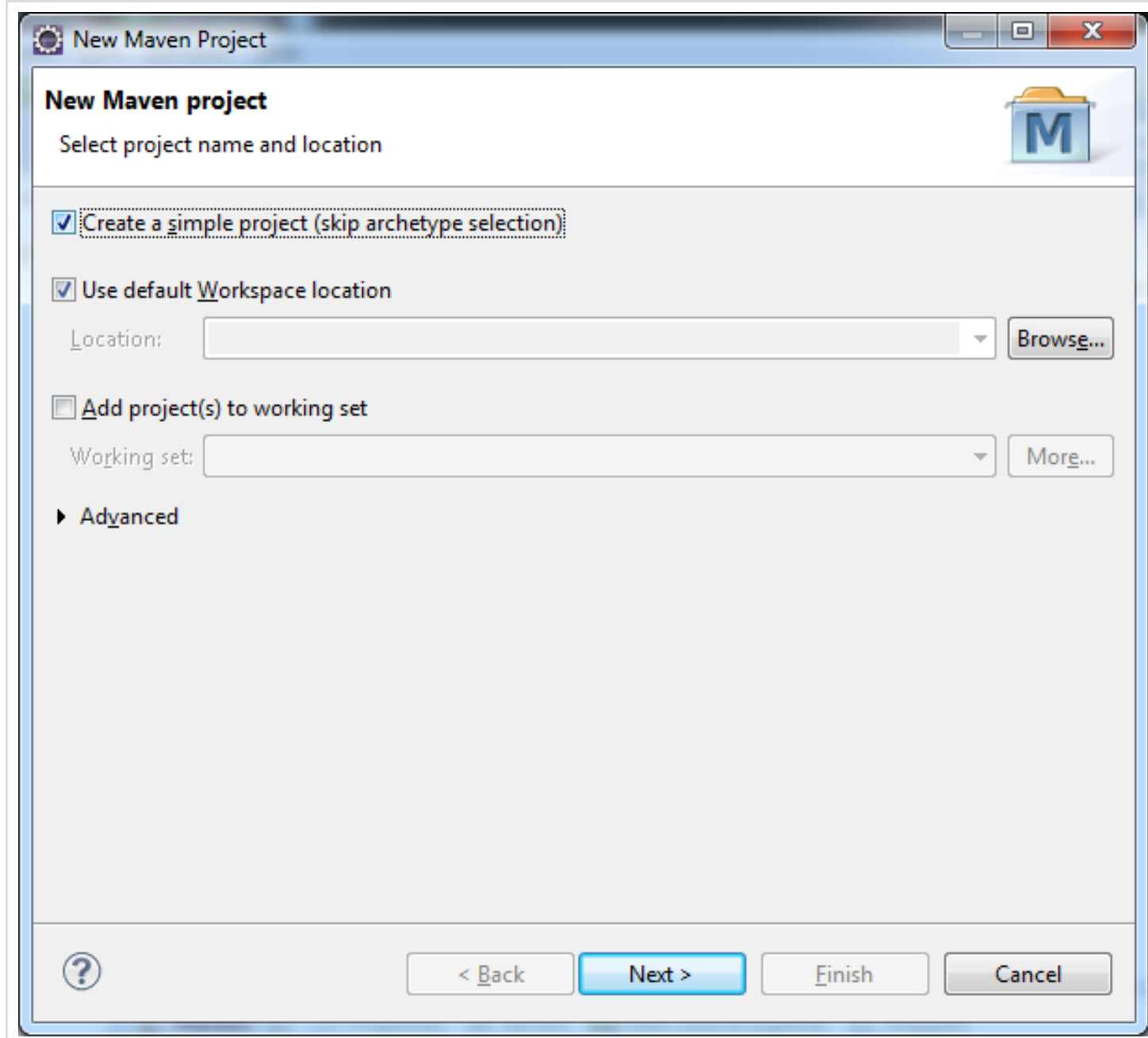
Vá até o menu `User Settings`. Há um *warning* lá dizendo que a configuração do usuário não existe. Você pode criar um outro `settings.xml` na pasta indicada ou simplesmente use um artifício (que eu sempre uso), que é definir o mesmo arquivo da configuração global.

(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/03-maven-eclipse.png>)

Caso não tenha entendido, o Maven possui um arquivo de configuração global que afeta diretamente a instalação e fica na pasta `conf`. Entretanto, cada usuário do sistema pode ter um arquivo próprio e sobrescrever as configurações globais que ele desejar. No entanto, se você é o único usuário do computador, não é necessário ter os dois arquivos.

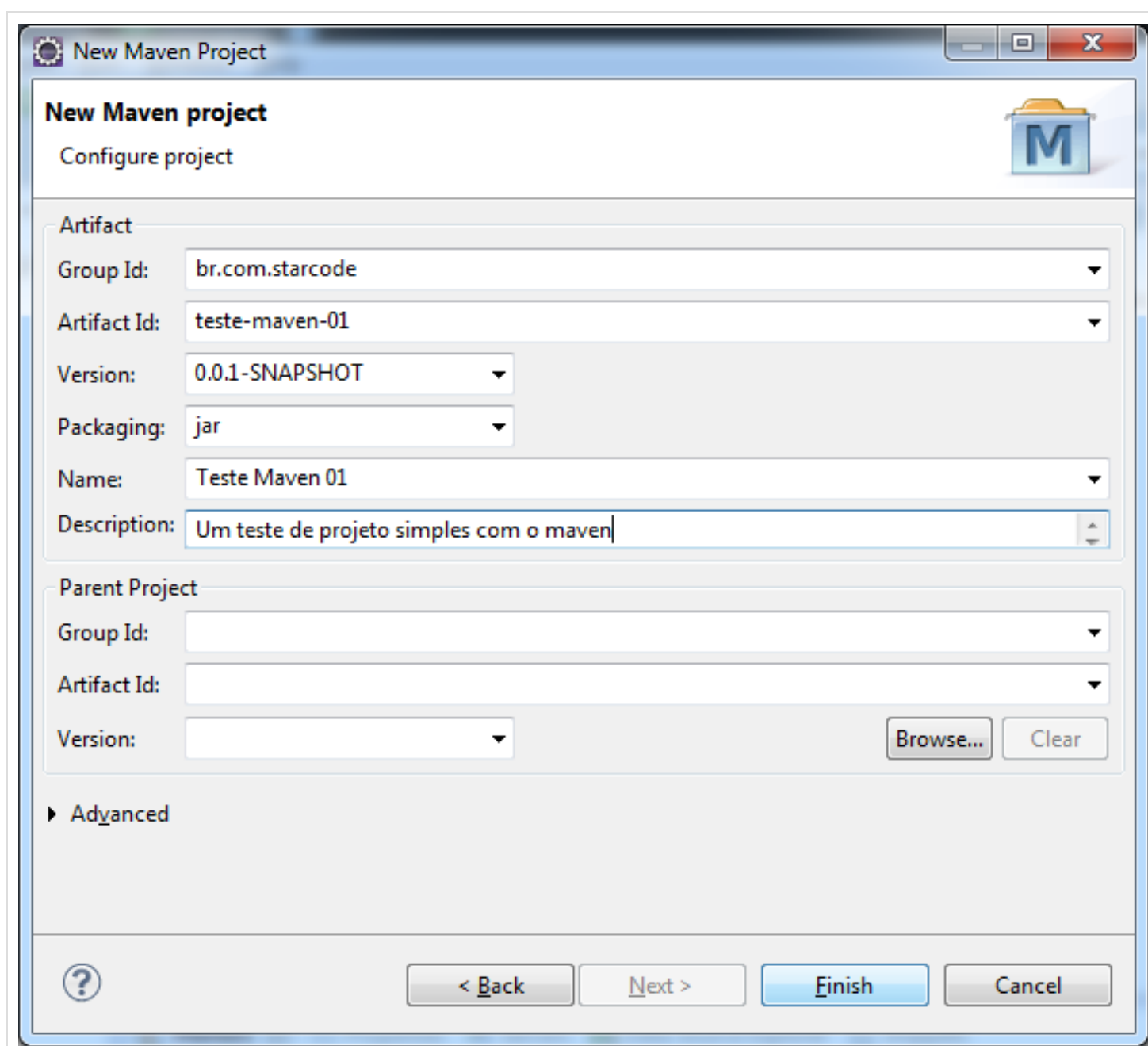
## Criando um projeto Maven simples no Eclipse

Com o Maven configurado, vamos criar um novo projeto no Eclipse. Acesse o menu `File > New > Maven Project`. Selecione a opção `Create a simple project (skip archetype selection)` e clique clique em `Next >`.



(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/01-projeto-simples.png>)

Vamos preencher a identificação do projeto, que nada mais é do que a identificação de um artefato.



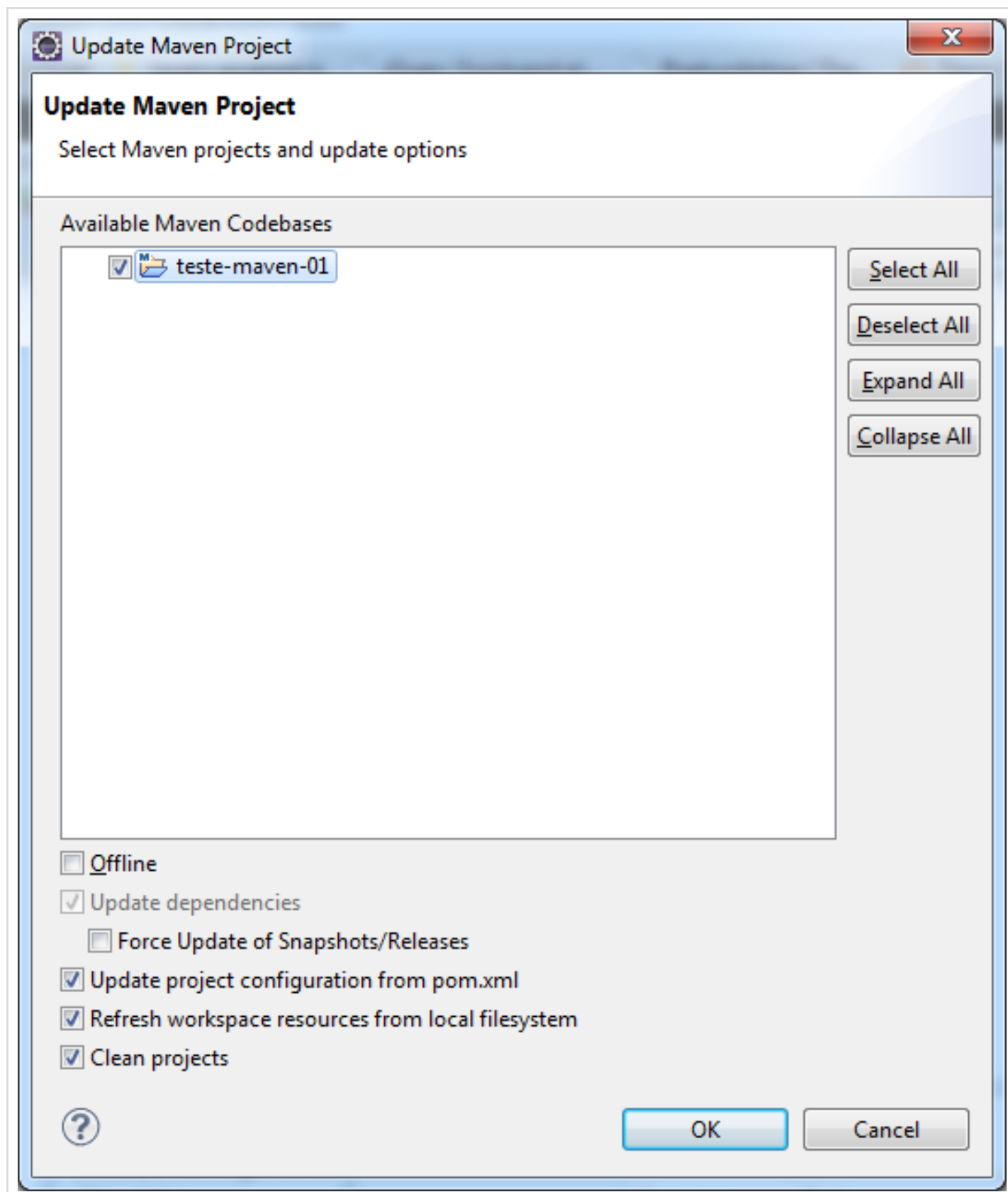


(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/02-projeto-simples.png>)

O Group Id para o exemplo será `br.com` (<http://br.com>).starcode 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.

Note que ele ainda não está definido com as configurações de um projeto Java, então clique com o botão direito sobre o projeto, acesse o menu `Maven > Update Project...`



(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/03-projeto-simples.png>)

Clique em `OK` para atualizar o projeto com as configurações do Maven e agora temos a estrutura característica.

(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/04-projeto-simples.png>)

## Adicionando manualmente uma dependência

Agora vou ilustrar como podemos adicionar algumas dependências ao projeto. Acesse o site mvnrepository.com (<http://mvnrepository.com/>), que contém um índice das dependências disponíveis no repositório do Maven. Pesquise por `commons-lang`.

(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/05-projeto-simples.png>)

Selecione o item `Apache Commons Lang`, como indicado na imagem abaixo:

(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/06-projeto-simples.png>)

Clique sobre a última versão (`3.3.2` na data em que escrevo o artigo).

(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/07-projeto-simples.png>)

Selecione e copie a identificação do artefato, conforme a imagem abaixo:

(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/08-projeto-simples.png>)

Agora volte ao Eclipse e clique duas vezes sobre o arquivo `pom.xml` para editá-lo. Provavelmente o editor foi aberto no aba `Overview` (veja abaixo do editor) com diversos campos e informações sobre o projeto.

Clique na aba `pom.xml` para mostrar o código fonte.

(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/09-projeto-simples.png>)

Adicione a tag `<dependencies>` logo abaixo da tag `<description>` e cole o conteúdo do site dentro dela.

**Dica:** Pressione `CTRL+A` para selecionar todo o conteúdo do arquivo e depois `CTRL+I` para indentar (tabular) o arquivo.

(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/10-projeto-simples.png>)

O conteúdo do `pom.xml` deve ser o seguinte:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
(http://maven.apache.org/POM/4.0.0)" xmlns:xsi="http://www.w3
.org/2001/XMLSchema-instance
(http://www.w3.org/2001/XMLSchema-instance)"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
(http://maven.apache.org/POM/4.0.0) http://maven.apache.org/x
sd/maven-4.0.0.xsd (http://maven.apache.org/xsd/maven-4.0.0.x
sd)">
    <modelVersion>4.0.0</modelVersion>
    <groupId>br.com.starcode</groupId>
    <artifactId>teste-maven-01</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>Teste Maven 01</name>
    <description>Um teste de projeto simples com o maven</des
cription>
    <dependencies>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
            <version>3.3.2</version>
        </dependency>
    </dependencies>
</project>
```

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.

Confira a entrada Maven Dependencies na imagem a seguir:

(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/11-projeto-simples.png>)

Pronto! Agora você já pode usar qualquer classe da biblioteca Apache Commons Lang (<http://commons.apache.org/proper/commons-lang/>).



Fiz uma classe de exemplo (File > New > Other..., selecione Class), com o seguinte conteúdo:

```
package br.com (http://br.com).starcode.testemaven01;

import org.apache.commons.lang3.StringUtils;

public class ClasseDeTeste {

    public static void main(String[] args) {

        System.out.println(StringUtils.capitalize("luiz"));

    }

}
```

Executei o método main clicando com o botão direito sobre a classe, menu Run As > Java Application. Veja o resultado:

(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/12-projeto-simples.png>)

## Executando os passos (*goals*) do Maven

Vamos supor que estamos construindo uma nova biblioteca.

Precisaremos **testá-la** (*test*), **empacotá-la** (*package*) num jar e **distribuí-la** (*deploy*) para o uso de terceiros, não é mesmo? O Maven nos ajuda grandemente com esses passos naturais do ciclo de vida de um projeto.

Vamos usar nosso projeto de exemplo e criar uma classe utilitária chamada `SuperUtil`:

```
package br.com (http://br.com).starcode.testemaven01;

import org.apache.commons.lang3.StringEscapeUtils;

public class SuperUtil {

    /**
     * Possibilita exibir um texto contendo HTML no navegador
     sem ataques XSS.
     * @param html Entrada do usuário (pode ter HTML, mas não
     deve ser renderizado, somente exibido)
     * @return Texto sem possíveis tags HTML
     */
    public static String escapeHTML(String html) {
        return StringEscapeUtils.escapeHtml4(html);
    }

}
```

Veja no Eclipse:

(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/01-executando-com-maven.png>)

Vamos ainda criar um teste unitário para nossa classe, as primeiro temos que adicionar a dependência do JUnit ao nosso projeto. Para isso vá até o site mvnrepository.com (<http://mvnrepository.com>) e pesquise por `junit`. Vá até a última versão (<http://mvnrepository.com/artifact/junit/junit/4.11>), copie o trecho do XML e adicione na seção de dependências do seu `pom.xml`.

Adicione também a tag `<scope>test</scope>` à esta dependência, para informar ao Maven que ela somente será usada no teste. Sim, o Maven é “esperto” e não incluirá, por exemplo, o JUnit na pasta `WEB-INF/lib` de uma aplicação web.

Veja como ficou o `pom.xml`:

```

<project xmlns="http://maven.apache.org/POM/4.0.0
(http://maven.apache.org/POM/4.0.0)" xmlns:xsi="http://www.w3
.org/2001/XMLSchema-instance
(http://www.w3.org/2001/XMLSchema-instance)"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
(http://maven.apache.org/POM/4.0.0) http://maven.apache.org/x
sd/maven-4.0.0.xsd (http://maven.apache.org/xsd/maven-4.0.0.x
sd)">
    <modelVersion>4.0.0</modelVersion>
    <groupId>br.com.starcode</groupId>
    <artifactId>teste-maven-01</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>Teste Maven 01</name>
    <description>Um teste de projeto simples com o maven</des
cription>
    <dependencies>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
            <version>3.3.2</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.11</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>

```

Ao salvar o arquivo o Maven deve baixar o JUnit automaticamente.

Agora crie a classe de teste SuperUtilTest em src/test/java:

```

package br.com (http://br.com).starcode.testemaven01;

import org.junit.Assert;
import org.junit.Test;

public class SuperUtilTest {

    @Test
    public void escapeHTMLTest() {

        String escapedHtml = SuperUtil.escapeHTML("<script>al
ert(1);</script>");
        String expected = "&lt;script&gt;alert(1);&lt;/script
&gt;";
        Assert.assertEquals(expected, escapedHtml);

    }

}

```

Caso queira executar o teste já, clique com o botão direito sobre a classe e acesse o menu Run As > JUnit Test:

(http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/02-executando-com-maven.png)

Confira o resultado:

(http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/03-executando-com-maven.png)

Sucesso! 😊

Imagine agora que tenhamos criado diversas classes e métodos. Temos uma versão beta de nossa biblioteca.

Vamos testar o projeto usando o Maven. Clique no projeto com o botão direito e na opção `Run As > Maven test`:

(http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/04-executando-com-maven.png)

Na primeira execução o Maven vai baixar diversos plugins e dependências internas para a execução do projeto. Aguarde um pouco e confira o resultado de todos os testes do projeto:

(http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/05-executando-com-maven.png)

Ok, agora vamos gerar um JAR do projeto. Clique no projeto com o botão direito e na opção `Run As > Maven build...`. Isso é necessário porque não é uma opção pronta para o passo `package`. Vá até o campo `Goals` e digite `package`.

(http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/06-executando-com-maven.png)

Clique em `Run` e aguarde.

(http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/07-executando-com-maven.png)

Se você observar o log no *Console* notará que os testes foram executados. Lembra que eu disse que os passos anteriores sempre são executados? Espero que agora tenha entendido melhor.

Note a última linha antes das palavras `BUILD SUCCESS`. Ali está o caminho do Jar gerado. Ele foi gerado dentro da pasta `target` do projeto.

Selecione a pasta `target` e Pressione `F5` para atualizá-la. Abra-a clicando na seta à esquerda e confira:

(http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/08-executando-com-maven.png)

Vamos agora executar o `install`, isto é, instalar o jar no repositório local. Clique no projeto com o botão direito e na opção `Run As > Maven install`. Aguarde um pouco e veja o resultado:

(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/09-executando-com-maven.png>)

As duas últimas linhas antes da mensagem de sucesso demonstram os locais onde o jar e o seu arquivo POM foram instalados:

```
[INFO] Installing D:\starcode\workspaces\workspace_demo\teste-maven-01\target\teste-maven-01-0.0.1-SNAPSHOT.jar to
c:\starcode\apache-maven-3.2.1\repo\br\com\starcode\teste-maven-01\0.0.1-SNAPSHOT\teste-maven-01-0.0.1-SNAPSHOT.jar

[INFO] Installing D:\starcode\workspaces\workspace_demo\teste-maven-01\pom.xml to c:\starcode\apache-maven-3.2.1\repo\br\com\starcode\teste-maven-01\0.0.1-SNAPSHOT\teste-maven-01-0.0.1-SNAPSHOT.pom
```

Vamos abrir o diretório do repositório local e dar uma olhadinha:

(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/10-executando-com-maven.png>)

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.

## Analizando as dependências

Vamos aprender a analisar as dependências de um projeto.

Abra novamente o seu arquivo `pom.xml` e vá até a aba `Dependency Hierarchy`. Você deve estar vendo isso:

(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/01-maven-dependencies.png>)

Abaixo, mais um exemplo de dependências extraído da minha biblioteca T-Rex (<https://github.com/utl Luiz/t-rex>):

(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/maven-dependency-hierarchy.png>)

## Usando o Maven em linha de comando

Tudo o que fizemos anteriormente através da IDE pode ser feito via linha de comando. É importante entender isso porque quando o projeto for compilado em um servidor de Integração Contínua, por exemplo, ele não contará com as facilidades do plugin que usamos.

Irei ilustrar a seguir a execução do `goal install` via linha de comando.

O primeiro passo é abrir o `CMD` e ir até o diretório do projeto. Então basta digitar o comando `maven install`.

(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/11-executando-com-maven.png>)

Configura o resultado da execução:

(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/13-executando-com-maven.png>)

Pronto!

Explicando: nós adicionamos o Maven ao `path` do sistema, lembra? `maven` é o nome do executável do Maven e `install` é o *goal* que você deseja executar. Também há usar o executável `mvn`, que é apenas um atalho para evitar digitar mais caracteres.

Da mesma forma, poderíamos executar `mvn test` ou `mvn package` como fizemos nos tópicos acima via menu do Eclipse para ativar os respectivos *Goals*.

Um *goal* muito importante ainda não mencionado é o `clean`, que serve para limpar todos os arquivos gerados da pasta `target`. Ele é muito importante para limpar eventual “sujeira” de gerações anteriores.

Quando estiver tendo problemas estranhos ou for gerar uma versão “oficial”, sempre use comandos como `mvn clean install` ou `mvn clean deploy` para garantir uma geração “limpa”.

Note que você pode especificar vários *goals* simultaneamente para o Maven executar. Nos exemplos acima, o Maven vai primeiro limpar o projeto e depois executar o `install` ou o `deploy`.

## Passando parâmetros para o Maven

Há ainda situações onde precisamos ajustar a execução do Maven através de parâmetros.



Por exemplo, em certas ocasiões queremos gerar um jar ou war para testes, porém o *build* do Maven falha porque um teste unitário terminou em erro.

Para resolver essa situação sem excluir o teste unitário, é possível configurar o Maven para “pular” os testes com o seguinte comando:

```
mvn -DskipTests clean install
```

(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/14-executando-com-maven.png>)

Configura o resultado da execução:

(<http://luizricardo.org/wordpress/wp-content/upload-files/2014/06/15-executando-com-maven.png>)

O mesmo resultado pode ser obtido no Eclipse clicando com o botão direito sobre o projeto e indo no menu `Run As > Maven build...`, digitando `clean install` no campo `Goals` e selecionando a opção `Skip Tests`.

## Distribuindo seu 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.

Disponibilizar seu projeto no repositório central do Maven exige que seu projeto seja *opensource* e que você solicite as permissões necessárias. Caso você queira fazer isso, siga os passos disponíveis neste link (<http://Sonatype%20OSS%20Maven%20Repository%20Usage%20Guide>).

Já dentro de uma empresa geralmente se configura um servidor para conter um repositório com os artefatos da empresa. Existem aplicações gratuitas que podem ser usadas para isso, como o Artifactory ([http://www.jfrog.com/home/v\\_artifactory\\_opensource\\_overview](http://www.jfrog.com/home/v_artifactory_opensource_overview)) ou o Nexus (<http://www.sonatype.org/nexus/>).

As configurações necessárias para o *deploy* num repositório remoto estão fora do escopo deste artigo, mas existem várias referências disponíveis na web.

## Leitura adicional

Usar o Maven para desenvolver projetos pessoais é relativamente fácil e este artigo cobre o necessário para isso. Entretanto, não deixe de estudar o material oficial e, aos poucos, ir entendendo os mecanismos do Maven para usá-lo em projetos maiores:

- Site oficial do Maven (<http://maven.apache.org/>)
- Maven in 5 Minutes (<http://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>)
- Maven Getting Started Guide (<http://maven.apache.org/guides/getting-started/index.html>)

## Considerações finais

Usar o Maven pode ser confuso a princípio, mas traz diversos benefícios.

Em curto prazo ele ajuda você a gerenciar as dependências e organizar seus projetos.

Em médio prazo você poderá ter um controle muito bom de versões e *releases*, além de um padrão maduro para seus projetos.

Em longo prazo o Maven possibilitará a Integração Contínua de seus projetos. Será necessário um esforço considerável, mas seus projetos serão compilados e testados automaticamente. Com uma quantidade de testes adequada para garantir que as funcionalidades existentes continuam funcionando e as novas vão funcionar, você pode ter versões sendo geradas todos os dias ou a cada *commit*!

Nesse ponto, cruzamos com conceitos de TDD, *Agile* e outros. Mas é exatamente este o motivo pelo qual ferramentas de automação como o Maven são importantes.

Espero que o leitor tenha compreendido seu funcionamento básico e possa evoluir daqui em diante para um melhor aproveitamento da automação em benefício da qualidade e da agilidade.

◀ automação (<http://luizricardo.org/tag/automacao/>)

◀ integração contínua (<http://luizricardo.org/tag/integracao-continua/>)


◀ maven (<http://luizricardo.org/tag/maven/>)    ◀ tutorial (<http://luizricardo.org/tag/tutorial/>)




**Luiz Ricardo**  
(<http://luizricardo.org/author/utluizgmail-com/>)

Ex-engenheiro, arquiteto, agilista, artesão, agora dedica-se a desenvolver software bem.

### Share!

 Facebook 33 (<http://luizricardo.org/2014/06/instalando-configurando-e-usando-o-maven-para-gerenciar-suas-dependencias-e-seus-projetos-java/?share=facebook&nb=1&nb=1>)

 Reddit (<http://luizricardo.org/2014/06/instalando-configurando-e-usando-o-maven-para-gerenciar-suas-dependencias-e-seus-projetos-java/?share=reddit&nb=1&nb=1>)

Twitter (<http://luizricardo.org/2014/06/instalando-configurando-e-usando-o-maven-para-gerenciar-suas-dependencias-e-seus-projetos-java/?share=twitter&nb=1&nb=1>)

LinkedIn 26 (<http://luizricardo.org/2014/06/instalando-configurando-e-usando-o-maven-para-gerenciar-suas-dependencias-e-seus-projetos-java/?share=linkedin&nb=1&nb=1>)

Google+ (<http://luizricardo.org/2014/06/instalando-configurando-e-usando-o-maven-para-gerenciar-suas-dependencias-e-seus-projetos-java/?share=google-plus-1&nb=1&nb=1>)

Mais

ANTERIOR

Instalando e configurando o Eclipse Kepler no linux Ubuntu  
(<http://luizricardo.org/2014/06/instalando-e-configurando-o-eclipse-kepler-no-linux-ubuntu/>)

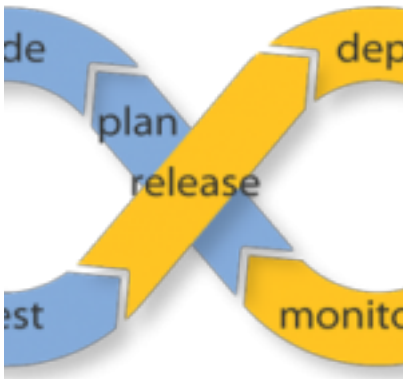
PRÓXIMA

Problemas de compatibilidade de versões no Java: especificando a versão do JDK para compilação  
(<http://luizricardo.org/2014/07/problemas-de-compatibilidade-de-versoes-no-java-especificando-a-versao-do-jdk-para-compilacao/>)

Related



(<http://luizricardo.org/2014/07/problemas-de-compatibilidade-de-versoes-no-java-especificando-a-versao-do-jdk-para-compilacao/>) Problemas de compatibilidade de versões no Java: especificando a versão do JDK para compilação  
(<http://luizricardo.org/2014/07/problemas-de-compatibilidade-de-versoes-no-java-especificando-a-versao-do-jdk-para-compilacao/>)



(<http://luizricardo.org/2015/05/deploy-e-integracao-continua/>) Deploy e Integração Contínua  
(<http://luizricardo.org/2015/05/deploy-e-integracao-continua/>)



(<http://luizricardo.org/2015/05/deploy-e-integracao-continua/>) Eclipse: acabando com alguns erros de validação desnecessários  
(<http://luizricardo.org/2013/10/eclipse-acabando-com-alguns-erros-de-validacao-desnecessarios/>)



(<http://luizricardo.org/2013/11/instalando-e-configurando-o-java-development-kit-7-para-desenvolvimento/>) Instalando e Configurando o Java Development Kit 7 para Desenvolvimento  
(<http://luizricardo.org/2013/11/instalando-e-configurando-o-java-development-kit-7-para-desenvolvimento/>)



([http://creativecommons.org/licenses/by-sa/3.0/deed.pt\\_BR](http://creativecommons.org/licenses/by-sa/3.0/deed.pt_BR)) O blog State of the Art

(<http://luizricardo.org>) de Luiz Ricardo (<http://luizricardo.org/sobre>) é licenciado sob uma Licença Creative Commons (<http://creativecommons.org/licenses/by-sa/3.0/deed>). Copie, compartilhe e modifique, apenas cite a fonte.