

Introdución a Maven

Maven é unha ferramenta de xestión de proxectos. Baséase nun ficheiro central, **pom.xml** (POM é o acrónimo de Project Object Model), onde se define todo o que necesita un proxecto. **Maven manexa as dependencias do proxecto, compila, empaqueta e executa os tests.**

Un dos **puntos fortes** de **Maven** é o mecanismo de xestión das **dependencias**.

Existen **tres** ferramentas de **automatización** de **compilación** de Java que teñen dominado, ata hoxe o ecosistema **JVM**: **Ant**, **Maven** e **Gradle**.

Apache Ant

Apache Ant é unha biblioteca de Java que se utiliza para automatizar os procesos de compilación de aplicacións Java.

O arquivo de compilación de Ant está escrito en **XML** e, por convención, denomínase **build.xml**.

As diferentes **fases** dun proceso de construción denomínanse **objectivos** (target).

A principal vantaxe de Ant é a súa flexibilidade. Ant non impón convencións de codificación nin estruturas de proxecto.

A falta de soporte integrado a xestión de dependencias, entre outras limitacións, levaron á creación de Maven.

Gradle

Gradle é unha **ferramenta de automatización de compilación e xestión de dependencias** que se baseou nos conceptos de **Ant** e **Maven**.

Gradle non usa arquivos XML.

Gradle trata de combinar a flexibilidade de Ant e as características de Maven.

Maven

Maven fíxose popular xa que os arquivos de compilación están estandarizados e leva menos tempo mantelos en comparación con Ant. Os arquivos de configuración de Maven, non obstante, tenden a ser grandes e difíciles de manexar.

As estritas convencións de Maven fano moito **menos flexible** que Ant.

A primeira vez que executemos maven, **creará un repositorio local no disco duro**. En concreto, creará o cartafol **.m2** no cartafol **home** do usuario. No repositorio local gardaranse todos os artefactos que manexe Maven.

O **cartafol predeterminado** do repositorio local é:

1. Unix/Mac OS X - ~/.m2/repository
2. Windows – C:\Users\{username}\.m2\repository

Grupos e artefactos

Un artefacto é un compoñente de software que podemos incluír nun proxecto como **dependencia**. Normalmente será un **jar**, pero podería ser doutro tipo, como un **war** por exemplo. Os artefactos poden ter dependencias entre si, por tanto, ao incluír un artefacto nun proxecto, tamén obteremos as súas dependencias.

Un grupo é un conxunto de artefactos. É unha maneira de organizalos.

Vexamos un exemplo:

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-api</artifactId>
  <version>5.6.0</version>
  <scope>test</scope>
</dependency>
```

Esta é a maneira de declarar unha dependencia do noso proxecto cun artefacto. **Indícase o identificador de grupo, o identificador do artefacto e a versión.**

Scope (alcance)

O scope serve para indicar o **alcance dunha dependencia** e a súa transitividade.

Hai dous tipos de dependencias en Maven: directa e transitiva.

As **dependencias directas** son as que se inclúen explicitamente no proxecto usando as etiquetas `<dependency>` no pom.xml.

As **dependencias transitivas** son as dependencias que requiren as nosas dependencias directas. As dependencias transitivas requiridas inclúense automaticamente no noso proxecto por Maven.

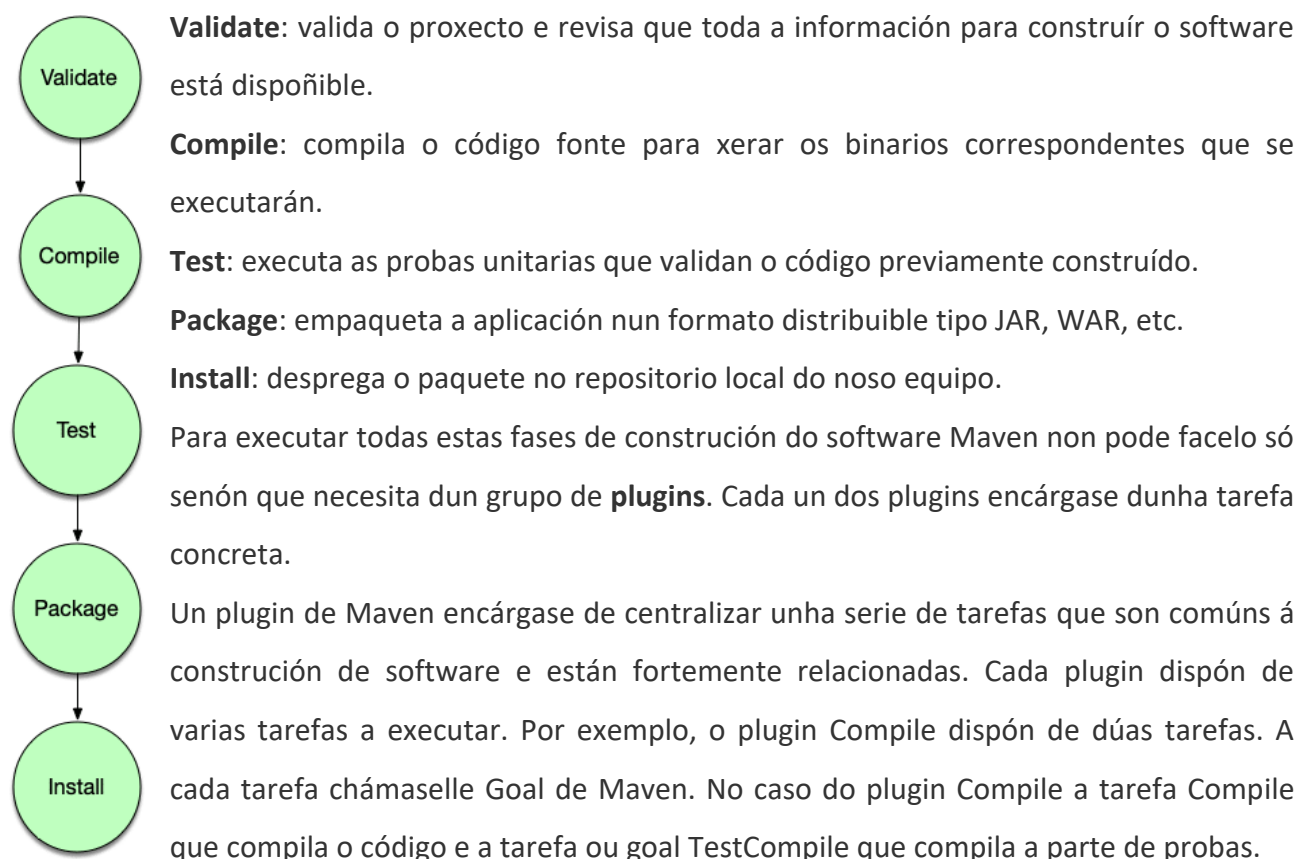
Hai 6 tipos:

- **compile**: é a que temos por defecto se non especificamos scope. Indica que a dependencia é necesaria para compilar. A dependencia propágase tamén nos proxectos dependentes.
- **provided**: é como a anterior, pero espera que o contedor ou o JDK xa teña esa biblioteca. Un claro exemplo é cando despregamos nun servidor de aplicacións que, por defecto, ten bastantes biblioteca que utilizaremos no proxecto, así que non necesitamos despregar a dependencia.
- **runtime**: a dependencia é necesaria en tempo de execución pero non é necesaria para compilar.
- **test**: a dependencia é só para testing que é unha das fases de compilación con Maven. JUnit é un claro exemplo disto. As dependencias de test non son transitivas e só están presentes para as rutas das clases de proba e execución.

- **system**: é como provided pero hai que incluír a dependencia explicitamente. Maven non buscará este artefacto no repositorio local. Haberá que especificar a ruta da dependencia no sistema mediante a etiqueta <systemPath>. Este alcance está obsoleto.
- **import**: este ámbito só está dispoñible para o tipo de dependencia pom. Indica que esta dependencia debe substituírse por todas as dependencias efectivas declaradas no POM.

Goals

Maven aporta unha serie de **cartafois por defecto** así como un **ciclo de vida** de construción do noso código. No ciclo de vida Maven define uns **pasos** moi concretos entre os que destacan:



Polo tanto, un Goal de Maven é unha tarefa que pertence a un plugin concreto e que pode usarse nunha fase do ciclo de vida de Maven.

Os plugins son artefactos que se inclúen no pom como <plugin>.

Os **plugins** poden ser de **compilación** ou de **informes**. Os plugins de compilación execútanse durante a compilación e debería configurarse como <build> no POM. Os plugins de informes executaríanse durante a xeración do sitio e deben configurarse como <report> no POM.

Arquetipo

Un **arquetipo**, traducido arquetipo é un modelo. Un arquetipo crea a estrutura do proxecto, o contido do pom.xml, a estrutura de cartafois e os ficheiros que inclúe por defecto.

Estrutura básica dun proxecto de Maven

A estrutura dependerá do tipo de proxecto co que traballemos, pero teñen cousas en común.

O **pom.xml** é onde se configura o proxecto Maven e o cartafol **src** é onde se colocan as fontes. Dentro de src, temos dous cartafois: **main**, onde vai todo o código do noso proxecto, e **test**, onde vai o código de probas. Nun proxecto java simple, dentro de main só temos o cartafol **java**. Nel xa podemos colocar os nosos paquetes e clases java.

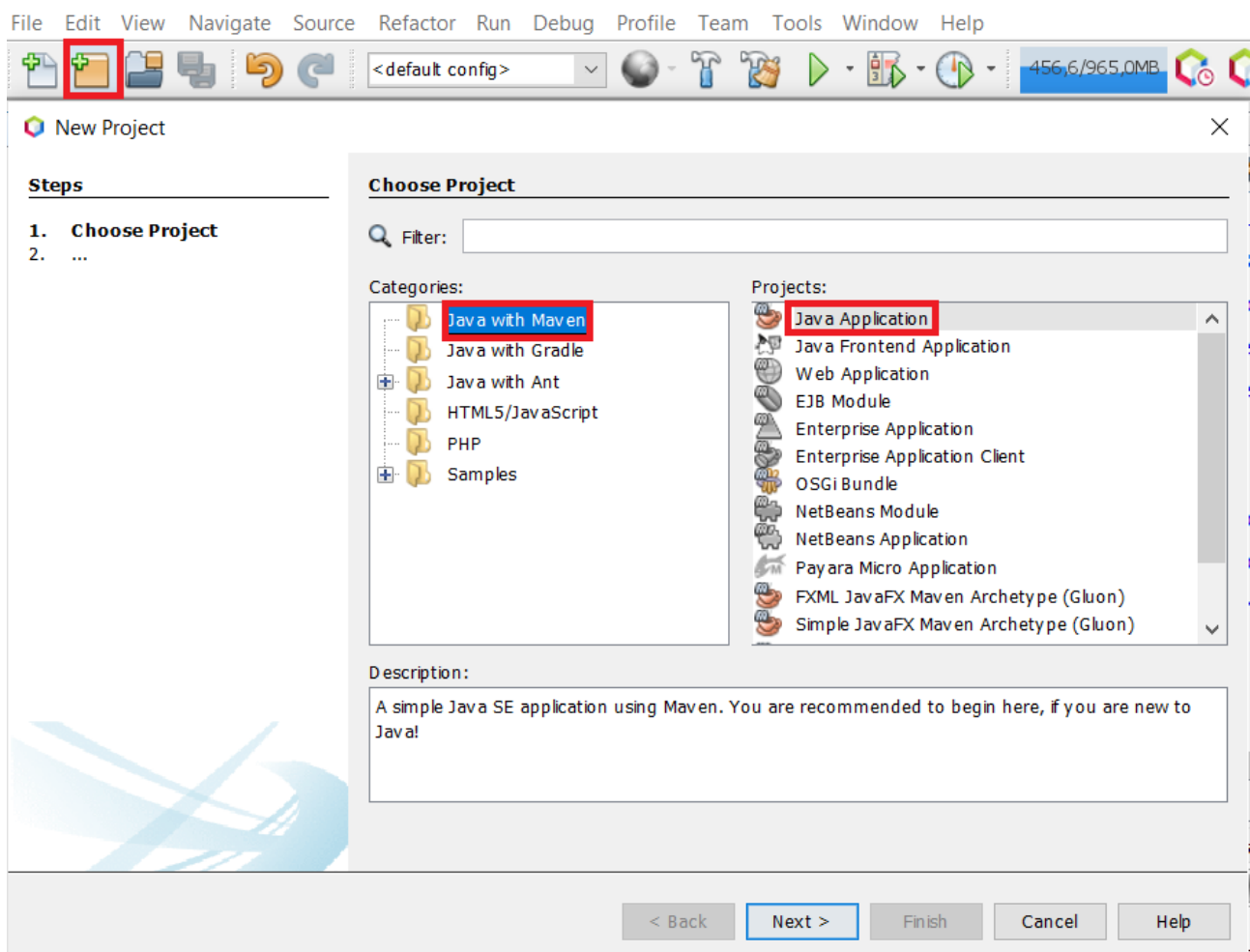
Proxectos Maven en Netbeans

Para crear un **novo proxecto Maven** podemos elixir a opción **New Project...** do menú **File** ou ben facer clic no **botón da barra de ferramentas** similar ao que aparece nese menú.

En función do tipo de distribución de NetBeans descargada e instalada, aparece unha listaxe de tipos de proxectos que se poden crear agrupados por categorías.

Para crear un programa básico en **Java** con **Maven** para ser utilizado na liña de comandos hai que seleccionar a categoría **Java with Maven** e o tipo de proxecto debe ser **Java Application**.

Tras facer clic no botón **Next>** comezará o asistente que permite crear un proxecto do tipo seleccionado.

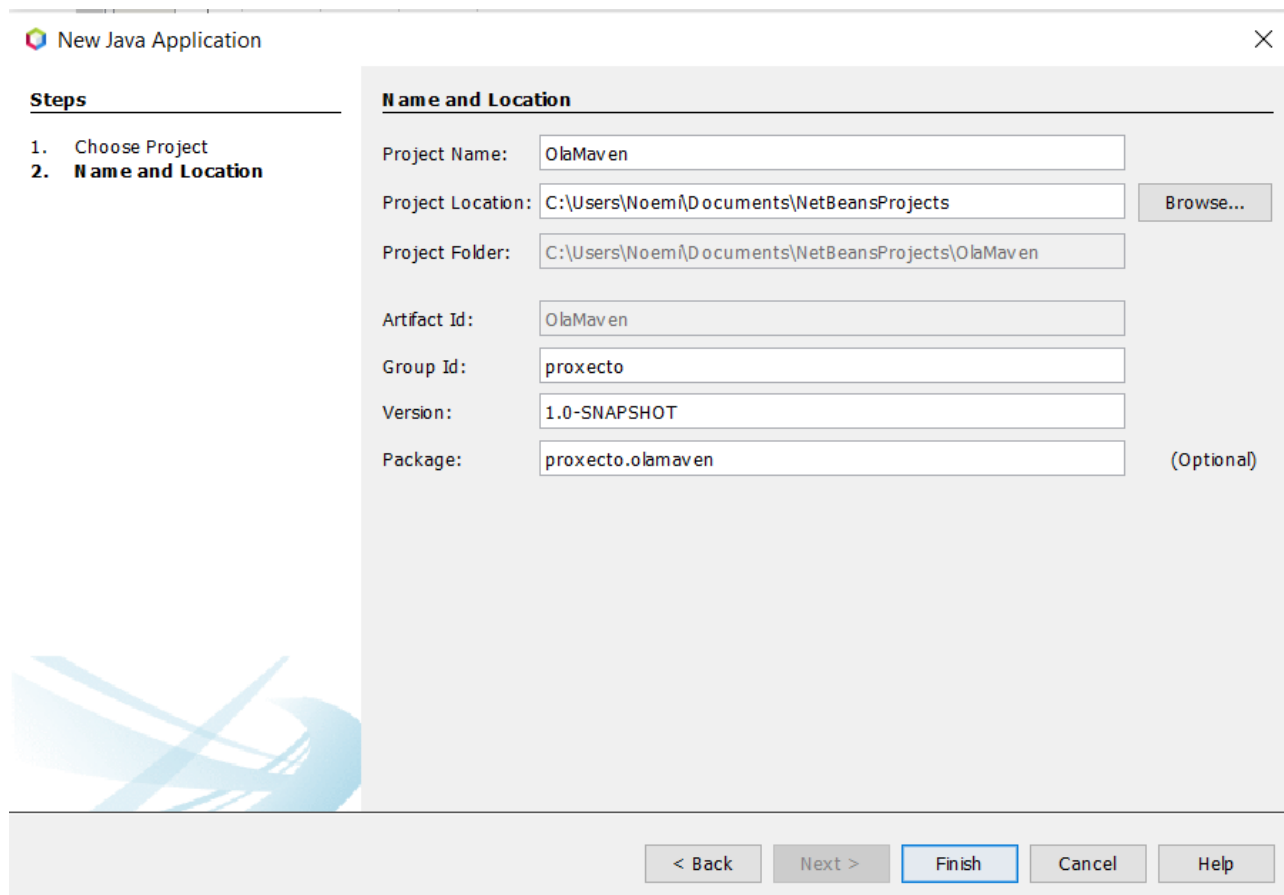


En primeiro lugar especificase o nome do proxecto a crear dentro do cadro de texto **Project Name**.

Para este exemplo indicamos o nome OlaMaven.

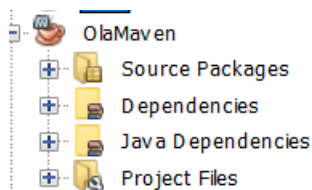
Se queremos modificar a ruta que aparece por defecto debemos utilizar o botón **Browse...** para especificar a nova ruta.

O resto de opcións permiten configurar o **artefacto** que se creará a partir deste proxecto.



Ao facer clic no botón **Finish** comezará o proceso de creación do novo proxecto.

Unha vez finalizado o proceso de creación do proxecto, abrírase automaticamente na ventá **Projects**. A lista de nodos dispoñible no proxecto depende do tipo de proxecto Maven creado. Neste caso créanse **Source Packages**, **Dependencies**, **Java Dependencies** e **Project Files** tal e como se pode ver na seguinte imaxe:



Dentro do nodo **Project Files** está o ficheiro **pom.xml** creado polo asistente **New Project**. O ficheiro pom.xml podemos abrílo facendo dobre clic sobre el na xerarquía do proxecto ou facendo clic dereito no proxecto e seleccionando a opción de menú contextual **Open POM**.

A ventá de edición do **pom.xml** de NetBeans permítenos ver un **grafo** de todos os artefactos usados no proxecto Maven. Isto pode ser moi útil cando tratamos de identificar todas as dependencias que existen nun proxecto.

Para engadir unha dependencia ao **POM**, podemos editar o POM directamente no editor ou abrir o menú contextual **Add Dependency** desde a ventá **Projects** ou desde o propio editor do POM. Expandimos a aplicación na ventá **Projects** e facemos clic no nodo **Dependencies** e eliximos **Add Dependency**.

Como exemplo imos engadir unha dependencia para traballar con métricas. As métricas poden proporcionar unha visión do funcionamento do código.

En **Query** escribimos **metrics-core** e en **Search Results** aparecerannos os resultados da busca e se eliximos un **jar** determinado xa se enchen automaticamente os campos correspondentes de **Group ID**, **Artifact ID** e **Version**.

Add Dependency

Group ID:

Artifact ID:

Version: Scope:

Type: Classifier:

Search **Open Projects** **Dependency Management**

Query:
(coordinate, class name, project name...)

Search Results:

- ☐ com.adobe.aam : metrics-core
- ☐ com.avast.metrics : metrics-core
- ☐ com.codahale.metrics : metrics-core
- ☐ com.flozano.metrics : metrics-core
- ☐ com.fnklabs : metrics-core
- ☐ com.jonny.matts.prometheus : metrics-core
- ☐ com.sinnerschneider.metrics : metrics-core
- ☐ com.tqdev.metrics : metrics-core
- ☐ com.yammer.metrics : metrics-core
- ☒ io.dropwizard.metrics : metrics-core

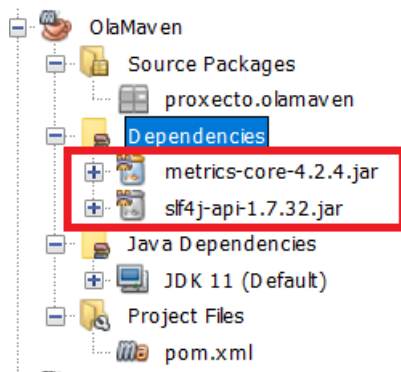
4.2.4 [jar] - central

Add **Cancel**

Tras pinchar no botón **Add** podemos ver que se engadiu a seguinte información no pom.xml:

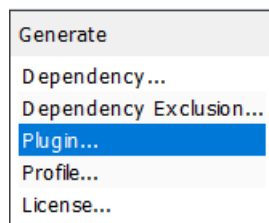
```
<dependencies>
  <dependency>
    <groupId>io.dropwizard.metrics</groupId>
    <artifactId>metrics-core</artifactId>
    <version>4.2.4</version>
  </dependency>
</dependencies>
```

Tamén vemos que aparecen os **jar** correspondentes no nodo **Dependencies**.

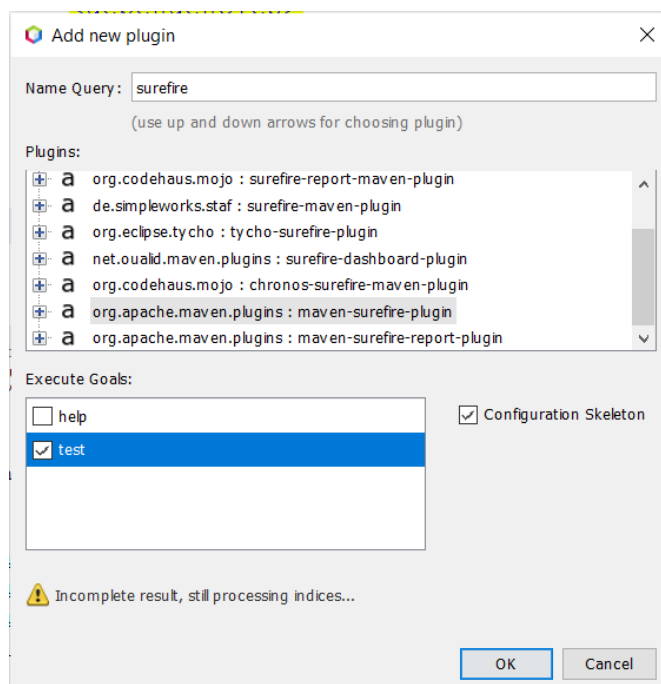


Para engadir un plugin ao POM clicamos no botón dereito e seleccionamos **Insert code...**

A continuación eliximos **Plugin...**



Imos engadir como exemplo o **Maven Surefire Plugin**.



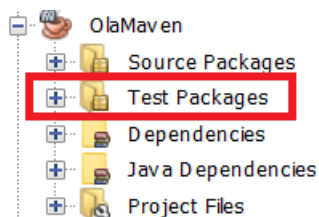
En **Name Query** facemos unha consulta poñendo surefire e seleccionamos o Maven Surefire Plugin.

Eliximos o goal ou goals (neste caso imos elixir test) e prememos no botón OK.

No **pom.xml** engádese automaticamente a seguinte información:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>3.0.0-M5</version>
      <executions>
        <execution>
          <goals>
            <goal>test</goal>
          </goals>
          <id>test</id>
        </execution>
      </executions>
      <configuration>
        <foo>bar</foo>
      </configuration>
    </plugin>
  </plugins>
</build>
```

No proxecto aparece un novo nodo Test Packages:



Se queremos executar tests unitarios nas nosas aplicacións xestionadas con **Maven**, podemos usar o plugin **Surefire**. Este plugin úsase durante a fase de test e xera informes en dous formatos: arquivo de texto plano (*.txt) e arquivo XML (*.xml). Por defecto os informes xeraranse no cartafol `${basedir}/target/surefire-reports`. O goal test encárgase de executar as probas.