



Índice

- Introducción:
 - Qué es Maven.
 - Principios de Maven.
 - Qué no es Maven.
 - Para qué sirve.
 - Características.
 - Característica de Maven versus Ant.
- **P**roject **O**bject **M**odel (POM)
 - Coordinadas de un proyecto Maven.
- Ciclo de vida del proyecto Maven.
- Estructura de directorios de un proyecto Maven

Índice

- Ciclo de vida del proyecto Maven.
- Estructura de directorios de un proyecto Maven
- Plugins y Goals
 - Repositorio de Maven.
- Tratamiento de Dependencias.
- Generación de site e informes del proyecto.

Qué es Maven

Maven, una palabra yiddish que significa acumulador de conocimientos.

Maven es una herramienta de software para la **gestión y construcción** de proyectos Java creada por Jason van Zyl, de Sonatype, en 2002. Similar en funcionalidad a Ant de Apache, pero tiene un modelo de configuración de **construcción** más simple, basado en un formato XML.

El proyecto Maven se inicio en un intento de simplificar el proceso de construcción del Yakarta Turbine Project (es un framework basado en los servlets que permite a los desarrolladores de Java crear rápidamente aplicaciones web.)

Hubo varios proyectos, cada uno con su propio archivos de generación Ant que eran un poco diferentes y JAR's registrados en el repositorio CVS.

Qué es Maven

Se pretendía obtener era:

- Una forma estándar de construir proyectos.
- Una definición clara de en qué consistía un proyecto.
- Una forma fácil de publicar la información de un proyecto.
- Una forma de compartir archivos JAR a través de varios proyectos.



El resultado es una herramienta que se utiliza para la **construcción y gestión** de cualquier proyecto basado en Java. Que **facilita** el trabajo del día a día de los desarrolladores de Java y ayuda a **comprender** mejor, cualquier proyecto basado en Java que utilice Maven.

Qué es Maven

Maven utiliza un **Project Object Model (POM)** para:

- Describir el proyecto de software a construir
- Incluir las **dependencias** con otros módulos y componentes externos.
- El orden de construcción de los elementos.

Como sea que en todo proyecto Java hay tareas que están claramente definidas, Maven las incorpora como son:

- La compilación del código.
- El empaquetado del código.

Qué es Maven

Maven es una herramienta open source para administrar proyectos de software. Por administrar, nos referimos a gestionar el ciclo de vida desde la creación de un proyecto en Java, hasta la generación de un binario que pueda distribuirse con el proyecto.

Maven nació dentro de la fundación Apache **para complementar a Ant**, la herramienta de compilación más usada en el mundo Java. Esto es, Ant permite crear scripts (usando XML) que indican cómo compilar un proyecto Java y generar un binario.

Qué es Maven

Maven complementa esta tarea brindándonos una estructura consistente de proyectos (todos los proyectos Maven tienen por defecto la misma estructura de directorios) y herramientas necesarias, para gestionar la complejidad de los proyectos de software: gestión avanzada de dependencias, informes sobre testing automáticos y extensibilidad vía plugins.

Maven sigue confiando en Ant para manejar la compilación e incluso puedes usar las tareas de Ant dentro de Maven.

Así que no es en sí un sustituto de esta herramienta, sino un complemento.

Maven Objetivos

El principal objetivo de Maven es permitir a un desarrollador comprender el estado completo de un proyecto y lo que queda de desarrollo en el menor período de tiempo.

Para ello hay varias áreas de preocupación que Maven intenta abordar:

- Que el proceso de **construcción** de un proyecto sea **fácil**.
- Que el sistema de **construcción** sea **uniforme**.
- Proporcionar **información de calidad** del proyecto.
- Proporcionar directrices para usar **las mejores prácticas** de desarrollo de software.
- Capaz de permitir la **migración** transparente a nuevas **características**.

Principios Maven Objetivos

Que el proceso de construcción de un proyecto sea fácil.

Maven no elimina la necesidad de conocer los mecanismos subyacentes , pero sin embargo proporciona un blindaje a la necesidad de conocer los detalles de los mismos.

Maven Objetivos

Un sistema de construcción de proyecto uniforme

Para construir un proyecto necesitamos:

- Proyecto Object Model (**POM**).
- Un conjunto de plugins que son compartidos por todos los proyecto Maven.

Estos dos componentes proporcionan un sistema de construcción de proyectos uniforme.

Una vez que estemos familiarizados con la forma en la que Maven construye **un** proyecto, conoceremos, automáticamente, como están contruidos **todos** los proyectos Maven.

Maven Objetivos

Proporcionar información de calidad del proyecto

Maven proporciona información útil del proyecto que es, en parte, obtenida de su **POM** y en parte generada a partir de los fuentes del proyecto.

Por ejemplo: Maven puede proporcionar:

- Cambios en el log de la documentación creado directamente de control del código fuente.
- Cruce fuentes de referencia.
- Listas de correo.
- Lista de dependencias.
- Informes de pruebas unitarias.

Maven Objetivos

Proporcionar la información del proyecto de calidad

Maven proporciona información mejor y transparente a los usuarios, por lo que de manera general mejorará el conocimiento general del proyecto lo que deberá redundar en la calidad del proyecto.

Otros productos pueden incluir plugins de Maven para proporcionar información adicional del proyecto.

Maven Objetivos

Proporcionar directrices para las mejores prácticas de desarrollo

Maven ayuda a desarrollar siguiendo los principios actuales de “buenas prácticas” del desarrollo de proyectos Java.

Por ejemplo: la especificación, ejecución y presentación de informes de las pruebas unitarias son parte normal del ciclo de construcción utilizando Maven.

Maven Objetivos

Proporcionar directrices para las mejores prácticas de desarrollo

Las **pruebas unitarias** hoy constituyen una de las mejores prácticas desarrollo de proyectos Java y en Maven se utiliza, con las siguientes directrices:

- Mantener el código fuente de las pruebas en un árbol separado, pero paralela.
- Usar los casos de test con las convenciones de nombres para localizar y ejecutar los test.
- Tener la configuración de los los casos de test en su entorno y no se basan en la personalización de la construcción de preparación de los test.

Maven Objetivos

Proporcionar directrices para las mejores prácticas de desarrollo

También ayuda en el flujo de trabajo del proyecto, como en la gestión de las release y en el seguimiento de bug.

Maven incluye guías sobre como se distribuyen la estructura de directorios de un proyecto Maven para, que una vez que aprenda la estructura, se puede navegar con facilidad por cualquier otro proyecto que utilice Maven.

Maven Objetivos

Permitiendo la migración transparente a nuevas características

Maven facilita la actualización de nuevas versiones. Así como la instalación de nuevos plugins de terceros o de Maven.

Qué no es Maven

Podríamos pensar que:

- Maven es una herramienta de sitios web y documentación.
- Maven extiende Ant para que pueda descargar dependencias.
- Maven es un conjunto de script de Ant reutilizables.

Maven cubre estas funcionalidades pero no son las únicas características que tiene Maven, y sus objetivos son bastante diferentes.

Cuando abordemos en un nuevo proyecto sin duda Maven es una buena elección, sin embargo para proyectos complejos existentes adaptarlos a Maven no siempre será posible.

Resumen de las características de Maven

- Configuración sencilla proyecto que sigue las mejores prácticas - Obtener un nuevo proyecto o módulo en pocos segundos
- Tratamiento homogéneo de todos los proyectos Maven, por lo que la curva de aprendizaje, al incorporar nuevos desarrolladores al proyecto, es mínima.
- Gestión de dependencias, incluyendo la actualización automática, cierres de dependencias (también conocido como dependencias transitivas)

Resumen de las características de Maven

- Capaz de trabajar fácilmente con varios proyectos al mismo tiempo.
- Extensible, con la capacidad para escribir fácilmente plugins en Java o lenguajes de scripting
- Es capaz de utilizar las tareas Ant para la gestión de las dependencias y el despliegue exterior de Maven

Resumen de las características de Maven

- Basado en Modelos:

Maven es capaz de construir cualquier número de proyectos con los tipos de salida predefinidos de JAVA : jar, war y ear, etc.

La distribución se basa en metadatos del proyecto, sin la necesidad de hacer ninguna de secuencias de comandos en la mayoría de los casos.

Resumen de las características de Maven

- Mantener un site con la información actualizada del proyecto:

Con el uso de los mismos metadatos para el proceso de construcción, Maven es capaz de generar un site o PDF incluyendo:

- Cualquier documentación que se quiere añadir
- Informes estándar, que sobre el estado del proyecto, genera Maven.

- Gestión y publicación de distribución de notas: Sin configuración adicional.

Resumen de las características de Maven

- Maven se integra con el sistema de control de versiones tales como CVS, Subversion o Git (actualmente se está en proceso de migración) y, además gestiona las release de un proyecto basado en una determinada etiqueta (tag).

[proyecto de migración de componentes SVN a Git](#)

- También puede publicar, en un lugar de distribución, para su uso por otros proyectos. Maven es capaz de publicar salidas individuales tales como un jar o un archivo que incluye otras dependencias y documentación, o una distribución de código fuente.

Resumen de las características de Maven

Gestión de la dependencia:

Maven fomenta el uso de un repositorio central de archivos JAR y de otras dependencias. Maven incorpora un mecanismo, para que los clientes de su proyecto, puedan descargar cualquier JAR que necesiten para la construcción de su proyecto.

Esto permite a los usuarios de Maven reutilizar JAR a través de proyectos y fomenta la comunicación entre los proyectos para asegurar que las cuestiones de compatibilidad con versiones anteriores se tratan.

Característica de un proyecto Maven

Maven mantiene un modelo de un proyecto:

- No es un simple compilador de código fuente.
- Está desarrollando una descripción de un proyecto de software y la asignación de un conjunto único de coordenadas para un proyecto. Usted está **describiendo** los atributos del proyecto.
 - ¿Cuál es la licencia del proyecto?
 - ¿Quién desarrolla y contribuye al proyecto?
 - ¿Qué otros proyectos tiene, de qué proyectos depende?
- Maven es algo más que una "herramienta de construcción".
- Es algo más que una mejora de otras herramientas como Ant.



Maven proporciona una base para el inicio, de una descripción semántica, de lo que consiste un proyecto de software.

Característica de un proyecto Maven

Manejo de dependencias

Un proyecto se define con coordenadas únicas que consiste en un **identificador de grupo**, **identificador de artefactos**, y la **versión**. Los proyectos pueden ahora utilizar estas coordenadas para declarar sus dependencias.

Repositorio remoto

Podemos usar las coordenadas definidas en el Project Object Model (POM) para crear repositorios de artifacts de Maven.

Característica de un proyecto Maven

Reutilización de la lógica de construcción

Los Plugins están codificados para trabajar con el POM; ellos no están diseñados para funcionar en archivos específicos en ubicaciones conocidas. Todo se abstrae en la configuración del modelo-plugin y la personalización sucede en el POM.

Integración con herramientas

Eclipse, NetBeans, IntelliJ tienen un lugar común para encontrar información sobre un proyecto. Antes de la llegada de Maven, cada entorno de desarrollo integrado (IDE) tenía una manera diferente para almacenar lo que era esencialmente un POM personalizado. Maven ha normalizado esta descripción, y aunque cada IDE sigue manteniendo los archivos de proyecto personalizado, puede ser generados fácilmente del POM.

Project Object Model.[POM]

Es la unidad fundamental de trabajo en Maven. Es un archivo xml. Reside **siempre** en el directorio base del proyecto y se llama pom.xml.

El POM contiene información sobre el proyecto y detalles de configuración utilizados por Maven para construir el proyecto.

POM también contiene los goals y plugins. Durante la ejecución de una tarea o goal, Maven busca el pom.xml en el directorio actual. Lee el pom.xml, obtiene la información de configuración que necesita, a continuación, ejecuta el goal.

Project Object Model.[POM]

Algunas de las configuraciones que se pueden especificar en el POM son los siguientes:

- Dependencias del proyecto
- Plugins
- Goals
- Construir perfiles
- Versión de proyecto
- Relación del equipos de los desarrolladores
- Lista de correo
- ...

Project Object Model. [POM]

```
pom.xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.ejrbalma</groupId>
  <artifactId>simple</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>simple</name>
  <url>http://maven.apache.org</url>

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

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Project Object Model. [POM]

```
<groupId>com.ejrbalma</groupId>  
<artifactId>simple</artifactId>  
<version>1.0-SNAPSHOT</version>  
<packaging>jar</packaging>
```



Los elementos:

- groupId
- artifactId
- version

se conoce como las coordenadas de Maven, que identifican de forma única un proyecto.

Project Object Model. [POM]

Antes de crear un POM, hay que decidir en primer lugar el grupo del proyecto (groupId), su nombre (artifactId) y su versión y sistema de versionado ya que estos atributos ayudan a identificar de forma exclusiva el proyecto en el repositorio.

Coordenadas GAV de un proyecto Maven

groupid: identificará de forma única su proyecto entre todos los proyectos, por lo que necesitamos cumplir un esquema de nomenclatura estricto.

Tiene que seguir las reglas de nombres de paquetes, lo que significa que tiene que ser por lo menos con un nombre de dominio que controles, y se puede crear tantos subgrupos como desee.

artifactId : Es el nombre del **jar** sin versión, se puede elegir cualquier nombre en **minúsculas** y sin símbolos extraños. Si es un desarrollo de terceros debemos utilizar el nombre *.jar

version: si el proyecto se distribuye, se debe elegir cualquier sistema de versionado de software típico con números y puntos (1.0, 1.1, 1.0.1, ...). Pero si se trata de un artefacto de un tercero, se tiene que usar su número de versión, sea el que sea y por extraño que parezca.

Coordenadas GAV de un proyecto Maven

groupid : Asocia un proyecto particular o conjunto de librerías con una compañía u organización.

Por convención se corresponde con la parte inicial de un paquete de Java usado para aplicaciones con clases

Ejemplos:

- org.apache.maven
- es.ulpgc.informatica.hpds
- com.ejrbalma

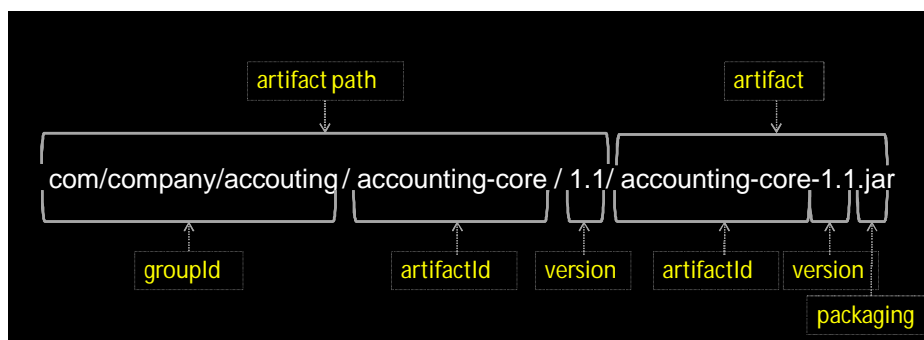
artifactId: representa el **nombre** del proyecto.

groupid+artifactId: Es el identificador **único** de un proyecto.

Coordenadas GAV de un proyecto Maven

Cada versión tiene su propio directorio en repositorio Maven, que es un subdirectorio del proyecto Maven.

com/company/accouting/accounting-core/1.1/accounting-core-1.1.jar



Coordenadas GAV de un proyecto Maven

- groupId
- artifactId
- version

Estos elementos se convierten en la clave de búsqueda y utilización de un proyecto particular en el vasto espacio de proyectos "Mavenized". Todos los repositorios Maven (públicos, privados, y locales) se organizan de acuerdo a éstos identificadores.

Cuando un proyecto se instala en un repositorio local de Maven, inmediatamente, a nivel local, se pone a disposición de cualquier otro proyecto. Lo único que debe hacer es añadirlo como una dependencia de otro proyecto, usando su coordenada Maven para un artifact específico.

Project Object Model. [POM]

```
<groupId>com.ejrbalma</groupId>
<artifactId>simple</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>
```



Los elementos:

- groupId
- artifactId,
- packaging
- versión

se conoce como las coordenadas de Maven, que identifican de forma única un proyecto.

```
<name>simple</name>
<url>http://maven.apache.org</url>
```



Nombre y url son elementos descriptivos del POM, proporcionando un nombre legible y asocia un sitio web con el proyecto.

Project Object Model. [POM]

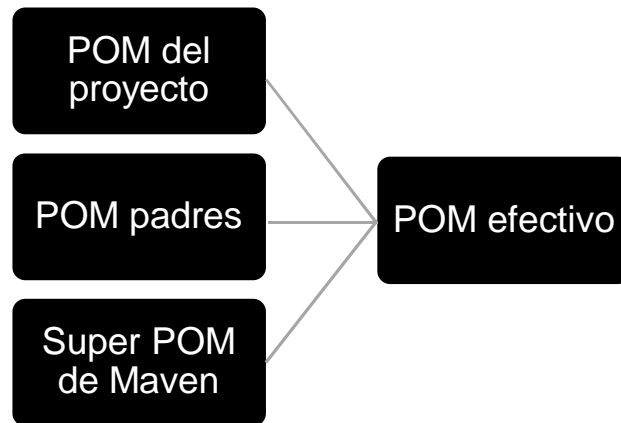
```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```



Las dependencias, en este caso define una dependencia, en el ámbito de test sobre framework de test unitarios Junit.

Todo proyecto Maven se ejecuta contra un POM efectivo, que es una combinación de ajustes entre el pom.xml del proyecto, todos los POM padres, un Super POM definido dentro de Maven, ajustes definidos por el usuario, y perfiles activos.

Project Object Model. [POM]



Plugin and Goal en Maven

Goal no es mas que un comando que recibe Maven como parámetro para que haga algo.

Para ejecutar un solo plugin y goal la sintaxis en Maven es:

\$ mvn plugin:goal

Por ejemplo:

\$ mvn archetype:generate

Donde:

archetype es el identificador del plugin
generate es el goal.

Maven tiene una arquitectura de plugins, para poder ampliar su funcionalidad, aparte de los que ya trae por defecto

Plugin and Goal en Maven

Ejemplos de goals serían:

\$ mvn clean:clean (o mvn clean): limpia todas las clases compiladas del proyecto.

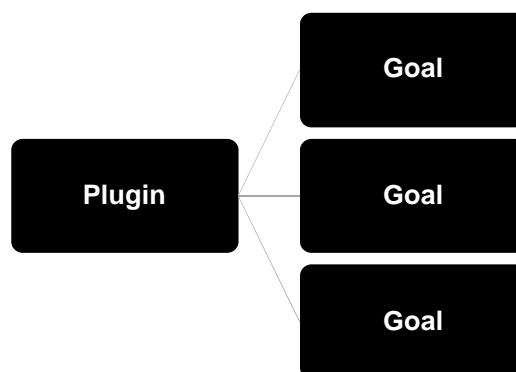
\$ mvn compile: compila el proyecto

\$ mvn package: empaqueta el proyecto (si es un proyecto java simple, genera un jar, si es un proyecto web, un war, etc...)

\$ mvn install: instala el artefacto en el repositorio local (/Users/home/.m2)

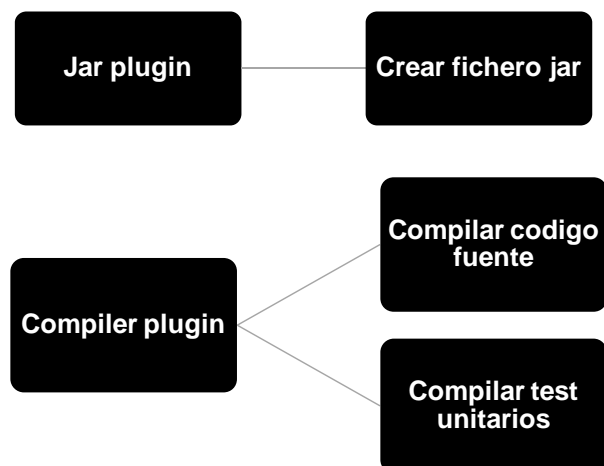
Plugin and Goal en Maven

Un plugin de Maven es una colección de uno o más goals.



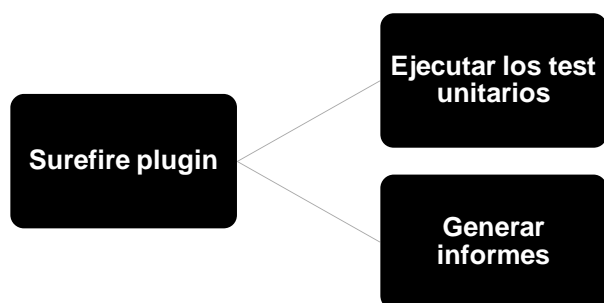
Plugin and Goal en Maven

Ejemplos de plugin-goals



Plugin and Goal en Maven

Ejemplos de plugin-goals



Todos estos plugin son del core de Maven, pero hay otros plugin más especializados. Como ...

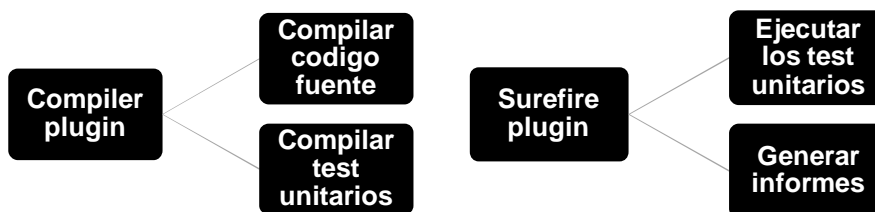
Plugin and Goal en Maven

Ejemplos de plugin-goals especializados

- Hibernate3, para la integración con la librería Hibernate.
- JRuby permite ejecutar ruby como una parte de un proyecto Maven o escribir plugin Maven en ruby.
- Maven también proporciona la habilidad para definir plugin personalizados.
 - Se pueden escribir en Java, pero también en otros lenguajes, incluyendo Ant, Groovy, beanShell, Ruby, ...

Plugin and Goal en Maven

Un goal es una tarea específica. El goal es la unidad de trabajo, ejemplos de goal:



Plugin and Goal en Maven

Los goals pueden ser parametrizados, configurando sus propiedades, que pueden ser usadas para personalizar su comportamiento.

Los goals pueden tener valores por defecto. Esto es una **convenciones** de Maven en la configuración.

Create goal:

Propiedad configurable : archetypeArtifactId
Valor por defecto: maven-archetype-quickstart

Genera un proyecto con una clase y con un fichero pom.xml

Si ejecutamos `$mvn archetype:create`

Plugin and Goal en Maven

Maven tiene poco que ver con las tareas específicas de construcción de un proyecto, Maven no sabe como compilar el código o incluso hacer un jar, Maven delega este trabajo en los plugin.

Cuando descargamos Maven nos bajamos su núcleo, que consiste en un shell que:

Sabe como analizar la línea de comandos.
Manejar classpath.
Analizar el archivo POM
Descargar plugin de Maven cuando sean necesarios.

Plugin and Goal en Maven

Al mantener el plugin del compilador separada del núcleo de Maven y disponer de un mecanismo de actualización, Maven hace que sea más fácil para los usuarios tener acceso a las últimas opciones del compilador.

De esta manera, plugins Maven permiten la reutilización universal de la lógica de construcción común. No se está definiendo la tarea de compilación en un fichero de construcción; está utilizando un plugin del compilador que es compartido por todos los usuarios de Maven.

Si hay una mejora en el plugin del compilador, cada proyecto que utiliza Maven puede beneficiarse inmediatamente de este cambio.

Si no te gustan el plugin para compilar, no hay problema, puedes reemplazarlo con tu propia implementación.

Repositorio de Maven

Los artifact y los plugins son descargados de un repositorio Maven cuando se necesitan. La descarga inicial de Maven apenas ocupa 1,5 MB y Maven se descarga los artifact y plugin a medida que los necesita de un repositorio remoto.

El repositorio remoto está [click](#).

Este repositorio se utiliza para:

- descargarse los plugin del core de Maven.
- Y las dependencias.

Repositorio de Maven

Si un proyecto depende de librerías que no son públicas ni libres, tendremos que configurar un repositorio personalizado dentro de la organización o descargar e instalar la dependencia de forma manual.

Los repositorios remotos por defecto pueden ser reemplazados o aumentados con los repositorios de la compañía. Existen productos disponibles que permiten mantener y gestionar espejo del repositorio Maven.

¿ Qué hace a un repositorio Maven ser un repositorio Maven?

Es una colección de artefactos almacenados en una estructura de directorios que coincide exactamente con las coordenadas de un proyecto Maven.

Repositorio de Maven

El estándar para un repositorio Maven es almacenar un artefacto en el directorio siguiente relativa a la raíz del repositorio:

```
/<groupId>\<artifactId>\<version>\<artifact>-\<version>.<packaging>
```

Los plugin y artifact son descargados del repositorio remoto y se almacenan en un repositorio local.

El repositorio local está ubicado:

```
C:\Users\USERNAME\.m2\repository
```

Maven utiliza el repositorio local para compartir dependencias entre proyectos locales.

Repositorio de Maven

Si trabajamos en dos proyectos A y B, supongamos que el proyecto B depende de algún artifact de A

Maven recuperará artifact del proyecto A de su repositorio local cuando se esté construyendo el proyectos B.

Los repositorio Maven:

- una caché local de artifacts descargados desde un repositorio remoto.
- un mecanismo para permitir que sus proyectos dependen unos de los otros.

Ciclo de vida de un proyecto Maven

El ciclo de vida de un proyecto Maven comienza con una fase de validación básica del proyecto y finaliza con una fase que incluye del despliegue del proyecto en producción.

Las fases del ciclo de vida son intencionadamente vagas, se definen únicamente como:

- Validaciones
- Pruebas
- Despliegue

Y pueden tener diferentes significados para diferente proyectos.

La fase package:

- Si es un proyecto java producirá un .jar
- Si es un proyecto web producirá un .war

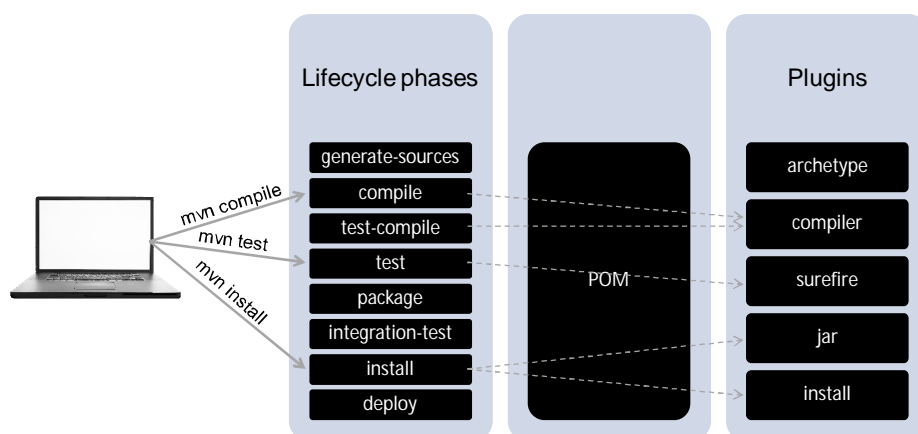
Ciclo de vida de un proyecto Maven

Goals de un plugin puede ser vinculados a una fase del ciclo de vida.

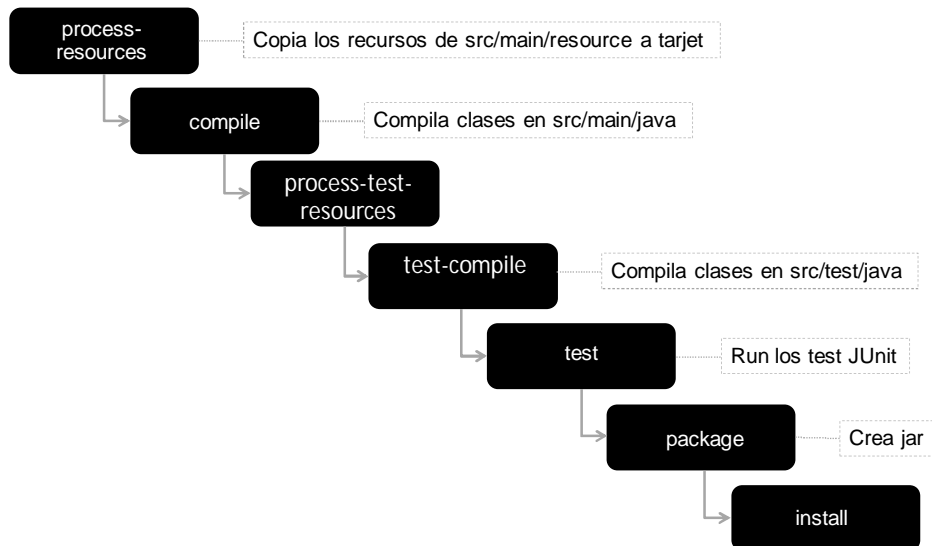
Maven se va moviendo entre las diferentes fases del ciclo de vida, ejecutará los goals vinculados a cada fase.

Cada fase puede tener asociados cero o mas goals asociados a ella.

Ciclo de vida de un proyecto Maven



Ciclo de vida de un proyecto Maven



Ciclo de vida de un proyecto Maven

Las partes principales del ciclo de vida de un proyecto Maven son:

- **generate-sources**: Genera código extra que necesite la aplicación, generalmente usando un adecuado plugins.
- **compile**: Genera los ficheros .class compilando los fuentes .java
- **test-compile**: Compila los test del proyecto.
- **test**: Ejecuta los test automáticos de JUnit existentes en src/test, abortando el proceso si alguno de los test falla. En todos los casos Maven genera informes en txt y xml en target/surefire-reports

Ciclo de vida de un proyecto Maven

Las partes del ciclo de vida principal del proyecto Maven son:

- **package:** Genera el fichero (.jar, .war, .ear) con los .class compiladas
- **integration-test:** Procesa y despliega los package si son necesarios en el entorno, en el cual, los test de integración puede ser ejecutados.
- **install:** Copia el fichero .jar a un directorio de nuestro ordenador donde Maven deja todos los .jar. De esta forma esos .jar pueden utilizarse en otros proyectos Maven en el mismo ordenador.
- **deploy:** Copia el fichero .jar a un servidor remoto, poniéndolo disponible para cualquier proyecto Maven con acceso a ese servidor remoto.

Ciclo de vida de un proyecto Maven

Cuando se ejecuta cualquiera de los comandos Maven, por ejemplo, si ejecutamos **mvn install**, Maven irá verificando todas las fases del ciclo de vida desde la primera hasta la del comando, ejecutando solo aquellas que no se hayan ejecutado previamente.

También existen algunos goals que están fuera del ciclo de vida que pueden ser llamados, pero Maven asume que estos goals no son parte del ciclo de vida por defecto (no tienen que ser siempre realizadas).

Ciclo de vida de un proyecto Maven

Otros goals son:

1. **clean**: Elimina todos los .class y .jar generados. Después de este comando se puede comenzar una compilación desde cero.
2. **assembly**: Genera un fichero .zip con todo lo necesario para instalar nuestro programa java. Se debe configurar previamente en un fichero xml qué se debe incluir en ese zip.
3. **site**: Genera un sitio web con la información de nuestro proyecto. Dicha información debe escribirse en el fichero pom.xml y ficheros .apt separados.
4. **site-deploy**: Sube el sitio web al servidor que hayamos configurado.
5. **Etc....**

Estructura de directorios de un proyecto Maven

Aunque en Maven la estructura del directorio de los proyecto Maven es una convención, sin embargo se puede modificar.

```
- src
- main
  + config
  + filters
  + java
  + resource
  + webapp
-site
-test
-target
  + clases
  + surefire-reports
  + test-classes
pom.xml
```

Motivos por los que mantener la estructura de directorios estándar de Maven:

- Un desarrollador que ha trabajado en proyectos Maven no tardará tiempo en acostumbrarse a la estructura de directorios de otro proyecto Maven.
- Simplifica el fichero PMO.
- Hace el proyecto fácil de comprender y facilitan la vida a la persona que debe de mantener el proyecto.
- Simplifica la integración de plugins

Dependencias en Maven

La administración de las dependencias en Maven es una de las características más potentes de Maven.

Las dependencias son las librerías que necesitamos para compilar, testear y ejecutar nuestra aplicación.

En otras herramientas como Ant las librerías se ubican en un directorio llamado /lib y en muchos casos son manejadas a mano.

En Maven se utiliza una aproximación **declarativa**.

Debemos declarar la librerías que necesita la aplicación, incluyendo el número de versión.

Maven encontrará, recuperará y ensamblará la librería que necesitamos en las diferentes partes del ciclo de vida del proceso de construcción de un proyecto Java.

Dependencias en Maven

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Las dependencias, en este caso define una dependencia, en el ámbito de test sobre framework de test unitarios Junit.

Dependencias en Maven

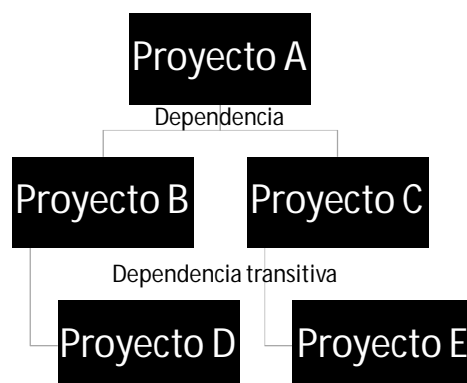
Supongamos que trabajamos en un proyecto que depende de una librería, y esta librería depende a su vez de otras librerías.

En lugar de rastrear todas las librerías una a una, en Maven declararemos una única dependencia y Maven localizará y se baja las otras librerías, esto es lo que se conoce como soporte de dependencias transitivas, siendo una de las características más potentes de Maven.

Maven también se encargará detectar los conflictos entre dependencias, y le ofrece la posibilidad de personalizar el comportamiento predeterminado y excluir ciertas dependencias transitivas.

Dependencias en Maven

Las dependencias de dependencias son llamadas dependencias transitivas.



Se pueden excluir dependencias transitivas y también es posible definir el scope de la dependencia.

Generar un site e Información en Maven

Otras de las características importantes de MAVEN es que es capaz de generar información e informes del proyecto, para ello solo tenemos que ejecutar:

```
$ mvn site
```

site es una de las fases del ciclo de vida de un proyecto Maven. Esta fase del ciclo de vida se refiere exclusivamente con el contenido del **síto** que se incluye bajo los directorios src/target/site y la generación de informes.

Después de ejecutar este comando deberíamos ver el web site en target/site. Cargando un target/site/index.html, deberíamos ver un web mínimo.

Generar un site e Información en Maven

Debe de contener:

Algunos reports en: "Project Reports "

Información del proyecto en: "Project Information"

- Dependencias.
- Desarrolladores.

En la medida de que el proyecto se vaya desarrollando al información proporcionado en la web irá siendo más rica y aportará más y mejor información al equipo de desarrollo.

Generar un site e Información en Maven

Hay informes por defecto:

Resultados de los test
Javadoc de la API del proyecto

Maven ofrece una amplia gama de informes que es personalizable:

- Clover: Examina la cobertura de la pruebas unitarias.
- JXR: Genera listados de código fuente HTML con referencias cruzadas útiles para las revisiones de código.
- PMD: analiza el código fuente para varios problemas de codificación.
- JDepend: analiza las dependencias entre los paquetes en un código.

Se pueden configurar los informes a través del POM.

Práctica 1 con Maven

Práctica 1 - Maven

1. Descargar, instalar y verificar la instalación Maven.
2. Obtener información de los plugin's de Maven.
3. Mi primer proyecto en Maven.
4. Determinar el POM efectivo
5. Creación de un site con información del Proyecto
6. Mi primer proyecto Maven en IntelliJ
7. Convertir un proyecto IntelliJ no Maven a Maven.
8. Analizar la estructura de directorios.
9. Ciclo de vida de un proyecto Maven – Ejecutar goals.

Instalación de Maven

Instalación de Maven en Windows

- Descargar la versión de Maven:
maven.apache.org/download
Seleccionar para la descarga : Binary zip archive, por ejemplo

Descomprimirlo en, por ejemplo:
ENTRAR/Mis Documentos/

Re-nombramos el directorio donde a descomprimido el zip a:
de apache-maven-3.3.9 a maven

Instalación de Maven

Y establecer dos variables de configuración:

Pre-requisito: tener en las variables de entorno

`JAVA_HOME = c:\Program Files\Java\jdk1.8.0_91`

`C:\> set M2_HOME=c:\Program Files\maven-X.X.X`

`C:\> set PATH=%PATH%;%M2_HOME%\bin`

Añadir a las variables de entorno para cuando arranque nuevamente el equipo

Verificar la instalación:

`$ mvn -version`

Instalación de Maven

Instalación OSX y Linux

`/usr/local % ln -s maven-2.0.9 maven`

`/usr/local % export M2_HOME=/usr/local/maven`

`/usr/local % export PATH=${M2_HOME}/bin:${PATH}`

También hay que modificar el fichero `.bash_config|profile`

Obtener ayuda en Maven

Maven Help plugin permite obtener una lista de los profiles activo en Maven, visualizar el efectivo POM, imprimir los setting efectivos o lista los atributos de un plugin de Maven.

La Ayuda de plugin de Maven tiene cuatro goals

Los tres primeros objetivos:

- active-profiles
- effective-pom
- effective settings

Describen un proyecto en particular y debe ser ejecutados en el directorio raíz del proyecto.

El último objetivo:

- describe

Es mas complejo, muestra información de un plugin o de un goal de Maven.

Obtener ayuda en Maven

Los siguientes comandos proporcionan información general a cerca de los cuatro objetivos

```
$ mvn help:describe -Dplugin=help
```

Información completa del plugin

```
$ mvn help:describe -Dplugin=help -Dfull
```

A vece solo necesitamos la información de un único objetivo.

```
$ mvn help:describe -Dplugin=compiler -Dmojo=compile -Dfull
```

```
$ mvn help:describe -Dplugin=archetype -Dmojo=generate -Ddetail
```

Pasos previos a la creación de un proyecto Maven

Antes de crear un proyecto Maven, hay que decidir en primer lugar el grupo del proyecto (groupId), su nombre (artifactId) y su versión ya que estos atributos ayudan a identificar de forma exclusiva el proyecto en el repositorio Maven (publico o privado).

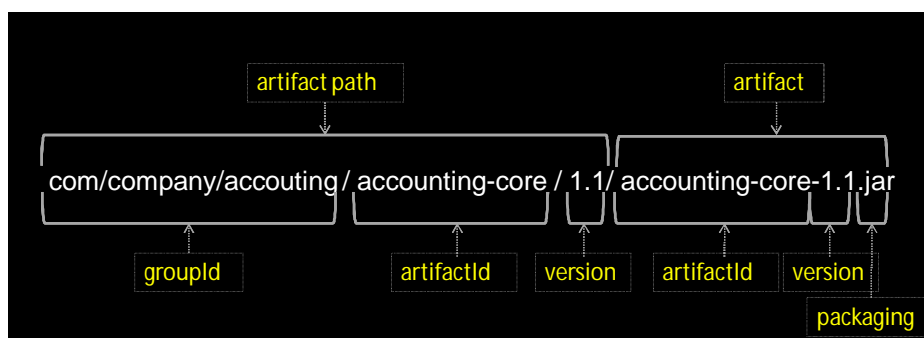
Coordenadas GAV del Proyecto Maven:

- groupId = **com.ulpgc.eii.hpds.ejrb**
- artifactId = **demosimple**
- Vversion = **1.0-SHAPSHOT**

Pasos previos a la creación de un proyecto Maven

Cada versión tiene su propio directorio en repositorio Maven, que es un subdirectorío del proyecto Maven.

com/company/accouting/accounting-core/1.1/accounting-core-1.1.jar



Pasos previos a la creación de un proyecto Maven

```
<groupId>com.ejrbalma</groupId>
<artifactId>simple</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>
```



Los elementos:

- groupId
- artifactId
- version
- packaging

se conoce como las coordenadas de Maven, que identifican de forma única un proyecto.

Crear un proyecto Maven desde consola

Abrir una consola y ejecutar:
Nos ubicamos en ..\Documents
\$ mvn archetype:genereta

874- version del plugin que nos genera el proyecto.
org.apache.maven.archetypes:maven-archetype-quickstart

Versión 6 (por ningún motivo en especial)

Define value for property 'groupId': : com.ulpgc.eii.ejrbalma
Define value for property 'artifactId': : demosimpleY
Define value for property 'version': 1.0-SNAPSHOT: :
Define value for property 'package': com.ulpgc.eii.ejrbalma: :jar

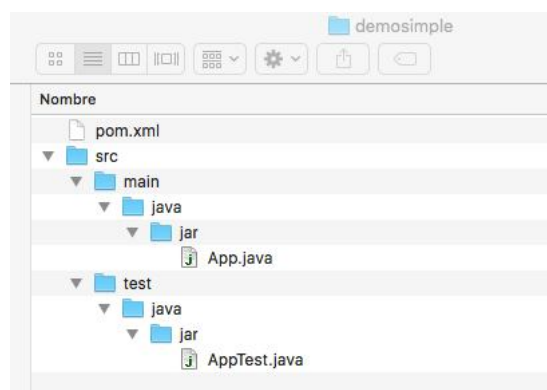
Y por último nos pide confirmación: Y

Crear un proyecto simple

Un archetype es una plantilla. Cuando creamos un proyecto, como hemos visto, tenemos que especificar uno. Un archetype crea la estructura del proyecto, el contenido del pom.xml, la estructura de carpetas y los ficheros que incluye por defecto.

Crear un proyecto simple

Crea una estructura de directorios tal cual:



Crear un proyecto simple

El contenido del fichero pom.xml generado es:

```
pom.xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.ejrbalma</groupId>
  <artifactId>simple</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>simple</name>
  <url>http://maven.apache.org</url>

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

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Crear un proyecto simple

Además crea una clase:

```
App.java
1  package jar;
2
3  /**
4   * Hello world!
5   *
6   */
7  public class App
8  {
9      public static void main( String[] args )
10     {
11         System.out.println( "Hello World!" );
12     }
13 }
14
```

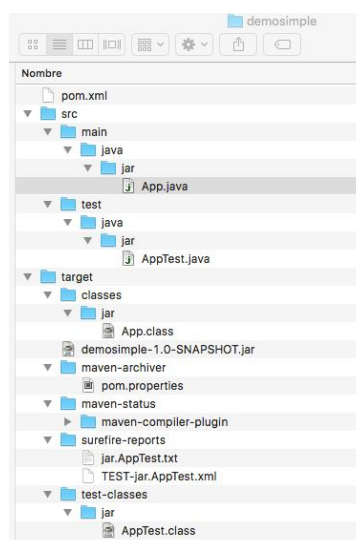
Crear un proyecto simple

El siguiente paso es construir un paquete de la aplicación. Para ello debemos estar situados en el directorio que contiene el fichero pom.xml y ejecutar el comando:

```
$ mvn install
```

Crear un proyecto simple

Crea una estructura de directorios tal cual:

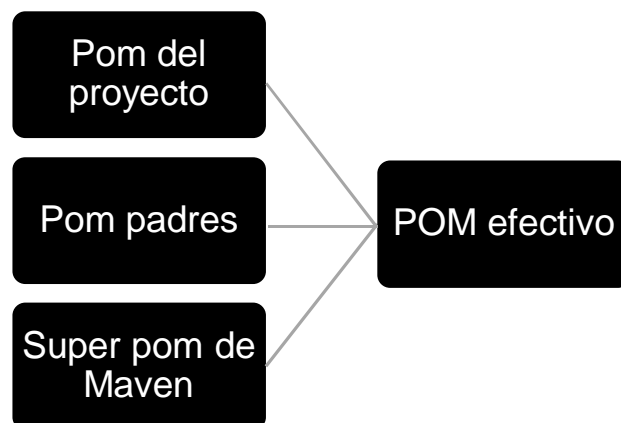


Crear un proyecto simple

Para ejecutar y verificar que se ha generado el paquete ejecutar el comando

```
java -cp target/demosimple-1.0-SNAPSHOT.jar  
com.ulpgc.eii.hpds.ejrbalma.demosimple.App
```

Crear un proyecto simple



Para ver este POM "eficaz", ejecute el siguiente comando en el directorio de base sencilla del proyecto:

```
$ mvn help:effective-pom
```

Generar un site e Información en Maven

Otras de la característica importantes de es que es capaz de Generar información e informes del proyecto, para ello solo tenemos que ejecutar:

```
$ mvn site
```

Site es una de las fases del ciclo de vida de un proyecto Maven. Esta fase del ciclo de vida se refiere exclusivamente con el contenido del sitio transformación bajo los directorios **src / site** y la generación de informes.

Después de ejecutar este comando deberíamos ver el web site en target/site. Cargando un target/site/index.html, deberíamos ver un web mínimo.

Pasos para crear un proyecto Maven en IntelliJ

- 1.- Abrir IntelliJ
- 2.- Create New Project.
- 3.- En el panel de la izquierda seleccionar Maven
- 4.- En el panel derecho, especifique las siguientes opciones:
 - 4.1.- Proyecto SDK - especifique su proyecto SDK (JDK).
 - 4.1.1. Si el JDK necesario ya está definido en IntelliJ IDEA, seleccionarlo de la lista.
 - 4.1.2.- De lo contrario, haga clic en Nuevo y seleccione la carpeta de instalación del JDK deseada.
 - 4.2.- Si desea crear un proyecto Maven basado en uno de los arquetipos Maven, seleccione la casilla de verificación Crear desde arquetipo, y seleccione el arquetipo deseado de la lista. Por ejemplo:
 - ✓ Create from archetype
 - ✓ Seleccionar: mave-archetyoe-quickstart
 - 4.3.- Puede hacer clic en Añadir Arquetipo y añadir un nuevo arquetipo a la lista de los ya existentes.
- 5.- Click en Next


Pasos para crear un proyecto Maven en IntelliJ

- 6.- Especificar las propiedades del proyecto:
 - 6.1.- Gruopld : com.ulpgc.eii.hpds.[ejrb] iniciales cada uno las suyas.
 - 6.2.- ArtifactId:demointellij
 - 6.3.- Version: 1.0-SNAPSHOT
- 7.- Click Next.
- 8.- Especificar:
 - Nombre del proyecto: demo2
 - Localización del proyecto: .../proyectosmaven/
- 9.- Click Finish [esperar a que se cree el proyecto...]

Pasos para crear un proyecto Maven en IntelliJ

Trabajando con las Herramientas de Maven.

Después de crear o importar un proyecto Maven IntelliJ visualiza el modelo Maven en la venta de herramientas de Maven. Podemos utilizar las herramientas Maven, para ejecutar los objetivos de Maven, crear configuraciones de ejecución/debug, mostrar las dependencias, personalizar la ejecución de objetivos, entre otras tareas.

- 1.- En el menú principal acceder a **View | Tool Windows | Maven Project** y abrimos la venta de herramientas maven.
- 2.- Veamos algunas de las tareas que podemos hacer:
 - 2.1.- Ejecutar metas (goals)
 - Click el icono Execute Maven Goal. 
 - Aparece una ventana e indicamos la meta a ejecutar o
 - Doble click en la meta directamente.

Pasos para crear un proyecto Maven en IntelliJ

Trabajando con las Herramientas de Maven.

2.2.- Crear una configuración de ejecución o debug para la lista de objetivos

- Marcar los objetivos que deseamos ejecutar.
- click en el botón derecho y seleccionar create “nombre de las metas” seleccionadas.
- En la venta que se abre, especificar configurar run/debug y establecer los parámetros y click OK.

2.3.- Visualizar Dependencias.

- click en el icono Show Dependencies



2.4.- Establecer triggers para la ejecución de metas.

- click en el botón derecho del ratón, sobre un goals y selecciona en que punto IntelliJ debería ejecutar dicha goal.
- Como resultado, el trigger para la ejecución del goal es visualizado la próxima vez que seleccionemos el goal.

Pasos para crear un proyecto Maven en IntelliJ

Trabajando con las Herramientas de Maven.

2.5.- Asignar un shortcut para ejecutar una goal.

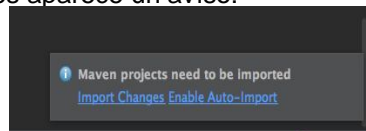
- Botón derecho en el goal y en el menú contextual seleccionar “Assign Shortcut”.
- En la ventana que se abre, Editar un shortcut añadir un nuevo shortcut y ok.

Pasos proyecto IntelliJ a IntelliJ Maven

1. Abrir un proyecto IntelliJ
 1. Clonar el siguiente proyecto:
 2. Abrir la consola git
 3. Crear un directorio ../Mis Documentos/proyecto
 4. Cambiar al directorio recién creado
 5. \$ git clone <https://bitbucket.org/ejrbalma/proyectintellij.git>
 6. Ejecutar el IntelliJ
 7. Open proyecto
 8. Localizarlo en la ubicación
2. Posicionamos a nivel de proyecto

Pasos proyecto IntelliJ a IntelliJ Maven

3. Seleccionar "Add framework Support..."
4. Seleccionar la tecnología deseada:
 - ☒ **m** Maven + OKEsperamos uno segundos
5. Se ha creado un fichero pom.xml y hemos de modificar como mínimo
 - groupId = Esta vacío, hemos de incluir nuestro
 - artifactId = nombre del proyecto
 - Version = 1.0-SNAPSHOT
6. En la parte inferior de la pantalla nos aparece un aviso:
7. Pulsamos sobre cualquier opción



Trabajando con un proyecto Maven en IntelliJ

Trabajando con un estructura de proyecto

Después de importar o crear un nuevo proyecto Maven vamos a verificar la estructura para confirmar que está ok.

- 1.- En el menú principal seleccionamos File | Project Structure
- 2.- En la ventana abierta, seleccionamos
- 3.- Seleccionamos la pestaña de dependencias.

Podemos hacer un download de librerías externas Maven en nuestro proyecto. Para ello:

- 1.- Seleccionamos en el panel de la izquierda "Libraries", click "+" y en la lista drop-down seleccionamos from Maven.

Se abre una ventana "Download Library from Maven Repository"

- 2.- Comenzamos a escribir el nombre de la librería + Pulsamos en buscar

Hibernate3

Trabajando con un proyecto Maven en IntelliJ

- 3.- Seleccionamos de la lista la librería que queremos y click en "OK"
- 4.- En la ventana que se abre, seleccionamos el módulo en el que queremos añadir la librería de Maven y "OK"

Como resultado se ha añadido la librería en la lista de dependencias. Además la librería se añadió a la ventana de nuestro proyecto en "External Libraries".

Trabajando con un proyecto Maven en IntelliJ

Trabajando con settings de Maven.

IntelliJ permite cambiar valores por defecto que son especificados en el proyecto actual. Bajo la categoría de Maven puede encontrar opciones que puede ayudar como configurar el proyecto.

Ejemplo, vamos a elegir trabajar en Work en modo offline, Se podría modificar el repositorio local, entre otros

1.- En el menú principal seleccionamos **File| Other Setting| Default Settings |Build, Execution, Deployed | Build Tools | Maven**

2.- En la venta que se abre especificamos los valores que deseamos para nuestro proyecto actual y **ok**

Explotando la potencia de Project Object Model

Trabajando con POM.

1.- Abrimos nuestro fichero POM en el editor y presionamos Alt+insert [menú principal Code|Generate].

2.- Aparece un lista con lo que podemos hacer, por ejemplo seleccionamos uno de ellos, "Dependency"

3.- En la ventana que se abre seleccionamos el artefacto que queremos añadir y pulsamos click.

Si no nos acordamos el nombre completo del artefacto podemos introducir parte del groupId y parte del artefacto separados por [:]

También podemos realizar la búsqueda por nombre de clase.

Configuración de Usuario y Repositorio local.

Una vez que empezamos a trabajar intensivamente con Maven, notaremos que tenemos en una ubicación de usuario un fichero de configuración y un repositorio local en nuestro directorio de trabajo. En ~/.m2 y tendremos:

settings.xml

Un fichero que contendrá configuración específica de usuario para autenticar, repositorios y otra información para personalizar el comportamiento de Maven.

repository/

Este directorio contiene el repositorio local de Maven. Cuando hacemos un download de una dependencia desde una ubicación remota Maven almacena una copia de la dependencia en el repositorio local.

Crear un proyecto simple

Cuando Maven se ejecuta, mira el contenido del Project Object Model para obtener la información del proyecto.

POM responde a preguntas como:

- ¿ Qué tipo de proyecto es?
- ¿Cuál es el nombre del proyecto?
- ¿ Hay personalización del proyecto?

El siguiente es el pom.xml más simple al que tendrá que hacer frente un proyecto Maven. Lo normal es que el pom.xml sea más complejo, con varias dependencias y con parametrización que re-defina el comportamiento de plugin.

Crear un proyecto simple

```
pom.xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.ejrbalma</groupId>
  <artifactId>simple</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>simple</name>
  <url>http://maven.apache.org</url>

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

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Crear un proyecto simple

```
<groupId>com.ejrbalma</groupId>
<artifactId>simple</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>
```

Los elementos:

- groupId
- artifactId,
- packaging
- versión

se conoce como las coordenadas de Maven, que identifican de forma única un proyecto.

```
<name>simple</name>
<url>http://maven.apache.org</url>
```

Nombre y url son elementos descriptivos del POM, proporcionando un nombre legible y asocia un sitio web con el proyecto.

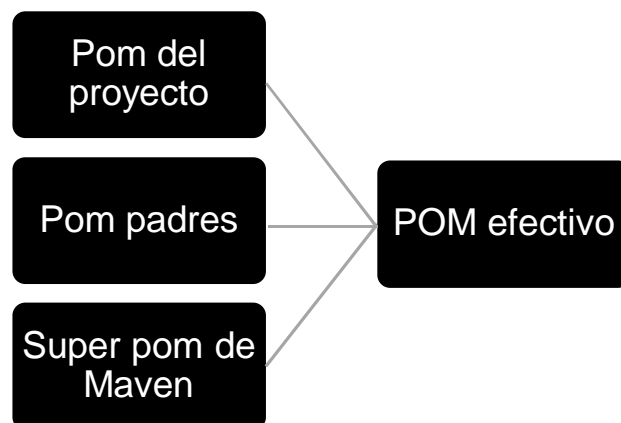
Crear un proyecto simple

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Las dependencias, en este caso define una dependencia, en el ámbito de test sobre framework de test unitarios Junit.

Todo proyecto Maven se ejecuta contra un POM efectivo, que es una combinación de ajustes entre el pom.xml del proyecto, todos los POM padres, un Super POM definido dentro de Maven, ajustes definidos por el usuario, y perfiles activos.

Crear un proyecto simple



Para ver este POM "eficaz", ejecute el siguiente comando en el directorio de base sencilla del proyecto:

```
$ mvn help:effective-pom
```

Dependencias en Maven

Usando la poderosa herramienta de Dependencias Transitivas, Maven no solo localizará la librería que hemos declarado, sino todas de las librerías que necesite la que hemos declarado.

Se declaran en la sección `<dependencies>` de `pom.xml` hemos de declarar la librería que tenemos que necesitamos compilar, test o run nuestra aplicación.

Las librerías serán recuperadas de los repositorios remotos o locales y cacheados en nuestra máquina local.

```
<dependencies>
  <dependency>
    <groupId>org.jfree</groupId>
    <artifactId>jfreechart</artifactId>
    <version>1.0.19</version>
  </dependency>
</dependencies>
```

Dependencias en Maven

Una librería puede tener diferentes versiones de librerías con el mismo número de versión por ejemplo misma versión pero para dos jdk distintos:

```
testing-5.1-jdk14.jar
testing-5.1-jdk15.jar
```

Cuando declaramos la dependencia hemos de indicar exactamente qué versión necesitamos, para ello Maven proporciona el elemento `<classifier>`

```
<dependencies>
  <dependency>
    <groupId>org.testing</groupId>
    <artifactId>testing</artifactId>
    <version>5.1</version>
    <classifier>jdk15</classifier>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Dependencias en Maven

Maven es flexible con el número de las versiones y podemos usar la notación de intervalos de la teoría de conjuntos.

```
(1,4)
[1,4]
[1,4)
[2,)
[1,5),(5,10]
```

```
<dependencies>
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate</artifactId>
    <version>[3.0,)</version>
  </dependency>
</dependencies>
```

Se requiere la última versión pero como mínimo la versión 3.0

```
<dependencies>
  <dependency>
    <groupId>commons-collection</groupId>
    <artifactId>commons-collection</artifactId>
    <version>[2.0,3.0)</version>
  </dependency>
</dependencies>
```

Ámbito de las dependencias

El *scope* sirve para indicar el **alcance de nuestra dependencia** y su transitividad. Hay 6 tipos:

compile: es la que tenemos por defecto sino especificamos scope. Indica que la dependencia es necesaria para compilar. La dependencia además se propaga en los proyectos dependientes.

provided: Es como la anterior, pero esperas que el contenedor ya tenga esa librería. Un claro ejemplo es cuando desplegamos en un servidor de aplicaciones, que por defecto, tiene bastantes librerías que utilizaremos en el proyecto, así que no necesitamos desplegar la dependencia.

runtime: La dependencia es necesaria en tiempo de ejecución pero no es necesaria para compilar.
en otro artículo sobre transitividad de dependencias.

Ámbito de las dependencias

Scope (alcance)

El *scope* sirve para indicar el **alcance de nuestra dependencia** y su transitividad. Hay 6 tipos:

test: La dependencia es solo para testing que es una de las fases de compilación con maven. JUnit es un claro ejemplo de esto.

system: Es como provided pero tienes que incluir la dependencia explícitamente. Maven no buscará este artefacto en tu repositorio local. Habrá que especificar la ruta de la dependencia mediante la etiqueta `<systemPath>`

import: este solo se usa en la sección *dependencyManagement*. Lo explicaré en otro artículo sobre transitividad de dependencias.

Una característica clave de Maven es que está listo para usar en red. El motor incluido en su núcleo puede dinámicamente descargar plugins de un repositorio, el mismo repositorio que provee acceso a muchas versiones de diferentes proyectos Open Source en Java, de Apache y otras organizaciones y desarrolladores. Este repositorio y su sucesor reorganizado, el repositorio Maven 2, pugnan por ser el mecanismo de facto de distribución de aplicaciones en Java, pero su adopción ha sido muy lenta.

Maven provee soporte no solo para obtener archivos de su repositorio, sino también para subir artefactos al repositorio al final de la construcción de la aplicación, dejándola al acceso de todos los usuarios. Una caché local de artefactos actúa como la primera fuente para sincronizar la salida de los proyectos a un sistema local.

Maven está construido usando una arquitectura basada en plugins que permite que utilice cualquier aplicación controlable a través de la entrada estándar.

En teoría, esto podría permitir a cualquiera escribir plugins para su interfaz con herramientas como compiladores, herramientas de pruebas unitarias, etcétera, para cualquier otro lenguaje.

En realidad, el soporte y uso de lenguajes distintos de Java es mínimo.