MAVEN

¿Por qué Maven?

Maven es una herramienta open-source, que se creó en 2001 con el objetivo de simplificar los procesos de build (compilar y generar ejecutables a partir del código fuente).

Es una herramienta capaz de gestionar un proyecto software completo, desde la etapa en la que se comprueba que el código es correcto, hasta que se despliega la aplicación, pasando por la ejecución de pruebas y generación de informes y documentación.

¿Por qué Maven?

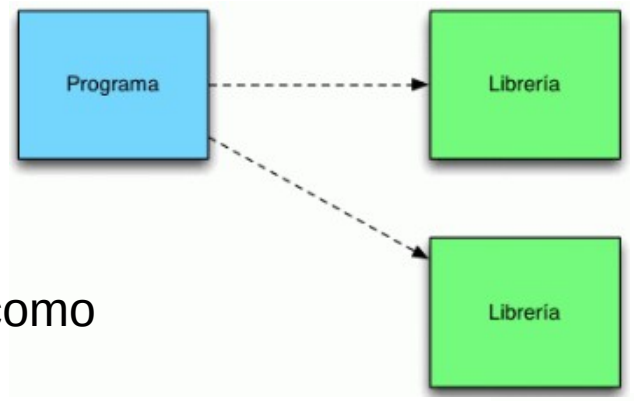
- Validación (validate): Validar que el proyecto es correcto.
- Compilación (compile).
- Test (test): Probar el código fuente usando un framework de pruebas unitarias.
- Empaquetar (package): Empaquetar el código compilado y transformarlo en algún formato tipo .jar o .war.
- Pruebas de integración (integration-test): Procesar y desplegar el código en algún entorno donde se puedan ejecutar las pruebas de integración.
- Verificar que el código empaquetado es válido y cumple los criterios de calidad (verify).
- Instalar el código empaquetado en el repositorio local de Maven, para usarlo como dependencia de otros proyectos (install).
- Desplegar el código a un entorno (deploy).

¿Por qué Maven?

Con Maven la gestión de dependencias entre módulos y distintas versiones de librerías se hace muy sencilla. En este caso, solo tenemos que indicar los módulos que componen el proyecto, o qué librerías utiliza el software que estamos desarrollando en un archivo de configuración de Maven del proyecto llamado POM.

Además, en el caso de las librerías, no es necesario descargarlas a mano. Maven posee un repositorio remoto (Maven central) donde se encuentran la mayoría de librerías que se utilizan en los desarrollos de software, y que la propia herramienta se descarga cuando sea necesario.

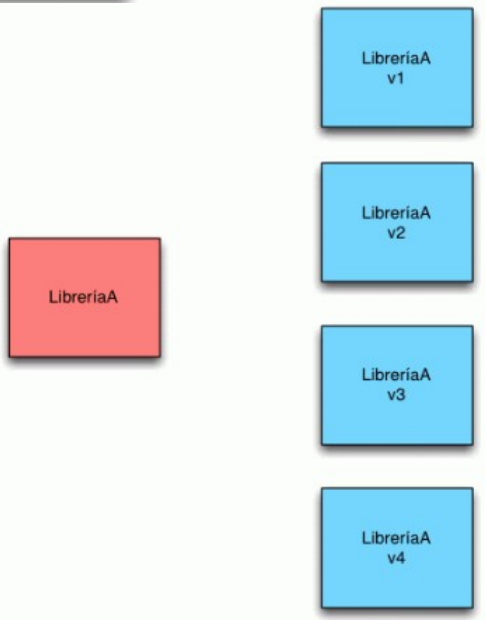
¿Qué es esto?



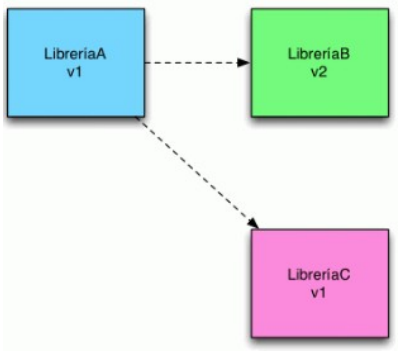
Normalmente cuando nosotros trabajamos con Java/JavaEE el uso de librerías es algo común como en cualquier otro lenguaje de programación.

Librerías y limitaciones

El concepto de librería es un concepto que a veces es limitado. Por ejemplo nosotros podemos querer utilizar la librería A en nuestro proyecto. Sin embargo no nos valdrá con simplemente querer utilizar la librería sino que ademas necesitaremos saber que versión exacta de ella necesitamos.



¿Es esto suficiente?. Lamentablemente no lo es, una librería puede depender de otras librerías para funcionar de forma correcta. Así pues necesitamos más información para gestionarlo todo de forma correcta.



¿Qué es esto?

Maven y Artefactos

Maven solventa esta problema a traves del concepto de Artefacto. Un Artefacto puede verse como una librería con esteroides (aunque agrupa mas conceptos). Contiene las clases propias de la librería pero ademas incluye toda la información necesaria para su correcta gestión (grupo, versión, dependencias etc).

Artefactos y POM

Para definir un Artefacto necesitamos crear un fichero POM.xml (Proyect Object Model) que es el encargado de almacenar toda la información que hemos comentado anteriormente:

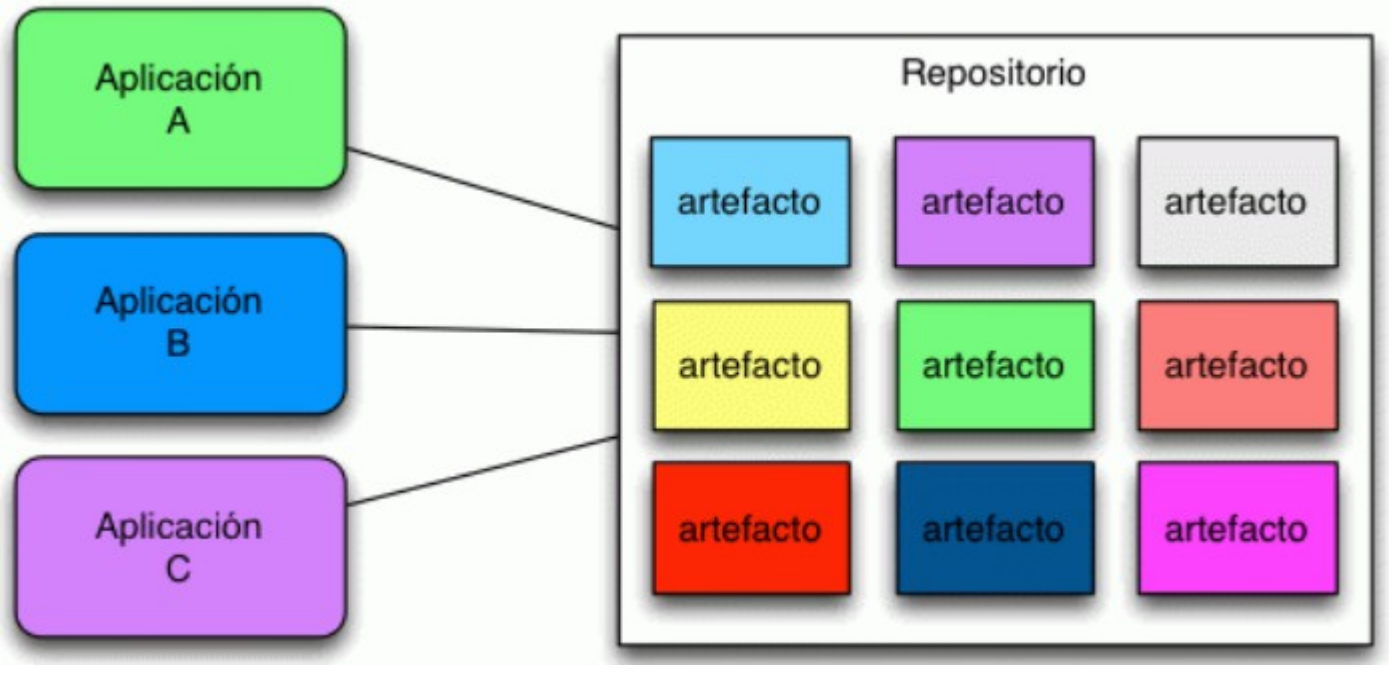

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

 <modelVersion>4.0.0</modelVersion>
 <groupId>com.genbetadev.proyecto1</groupId>
 <artifactId>proyecto1</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <packaging>jar</packaging>
 <dependencies>

 <dependency>
 <groupId>log4j</groupId>
 <artifactId>log4j</artifactId>
 <version>1.2.17</version>
 </dependency>

</dependencies>
</project>
```

La estructura del fichero puede llegar a ser muy compleja y puede llegar a depender de otros POM. En este ejemplo estamos viendo el fichero más sencillo posible. En el se define el nombre del Artefacto (artifactID) el tipo de empaquetado (jar) y también las dependencias que tiene (log4j). De esta manera nuestra librería queda definida de una forma mucho más clara.

¿Qué es esto?

Maven Repositorio y Artefactos

Una vez definidos correctamente todos los Artefactos que necesitamos, Maven nos provee de un Repositorio donde alojar, mantener y distribuir estos. Permitiéndonos una gestión correcta de nuestra librerías, proyectos y dependencias.



Uso desde línea de comando

Uso desde línea de comando

Para ver la manera de usar Maven desde línea de comando, se puede consultar la siguiente página:

Primeros Pasos con Maven

http://www.chuidiang.org/java/herramientas/maven.php

Nosotros lo usaremos desde la IDE, de todas formas haremos un resumen de algunos conceptos.

Uso desde línea de comando

Primeros pasos con Maven

En nuestros proyectos java siempre tenemos varias tareas que realizar. La primera suele ser crear una estructura de directorios para nuestro proyecto, con un hueco para los fuentes, otro para iconos, ficheros de configuración o datos, directorio para dejar los .class o el .jar, para dejar el javadoc, etc, etc.

Después, tenemos más tareas que realizamos con cierta frecuencia, como borrar los .class, compilar, generar la documentción de javadoc, el jar, incluso generar documentación web para publicar nuestro trabajo. Posiblemente acabemos haciendo algunos scripts o ficheros .bat para todas estas tareas.

Si nuestro programa es grande, incluso es posible que dependamos de otros jar externos, como drivers debase de datos, JUnit para clases de test, log4j para nuestra salida de log, etc, etc. Tendremos que copiar todos esto jar externos en algún sitio de nuestro proyecto e incluirlos.

Uso desde línea de comando

Una herramienta evolucionada para hacer esto es maven. Maven, con comandos simples, nos crea una estructura de directorios para nuestro proyecto con sitio para los fuentes, los iconos, ficheros de configuración y datos, etc, etc. Si a maven le indicamos qué jar externos necesitamos, es capaz de ir a buscarlos a internet y descargarlos por nosotros. Sin necesidad prácticamente de configurar nada, maven sabe como borrar los .class, compilar, generar el jar, generar el javadoc y generar un documentación web con montones de informes (métricas, código duplicado, etc). Maven se encarga de pasar automáticamente nuestros test de prueba cuando compilamos. Incluso maven nos genera un zip de distribución en el que van todos los jar necesarios y ficheros de configuración de nuestro proyecto.

Uso desde línea de comando

Acá viene el resumen…

- Instalar maven
- Crear un proyecto
  - mvn archetype:create -DgroupId=chuidiang.ejemplos -DartifactId=EjemploMaven
  - Una vez ejecutado este comando, Maven empezará a bajarse cosas de internet cuando lo ejecutemos por primera vez (en los próximos proyectos ya no necesita bajarse nada) y creará una esctructura de directorios y ficheros como la siguiente

```
Ejemplo Maven
+---src
¦   +---main
¦   ¦   +---java        //Para nuestros fuentes
¦   ¦       +---chuidiang
¦   ¦           +---ejemplos
¦   ¦               +---App.java
¦   +---test
¦       +---java        //Para test de Junit
¦           +---chuidiang
¦               +---ejemplos
¦                   +---AppTest.java
+---pom.xml
```

Uso desde línea de comando

Continuación del resumen…

Ahora llega el momento duro. Debemos empezar a escribir el código, tanto de nuestro proyecto como de las clases de test de Junit

- Compilar
  - **mvn compile** Esto creará un directorio target justo debajo de EjemploMaven y ahí un subdirectorio classes donde meterá todos los .class de nuestro compilado

```
EjemploMaven
+---src
¦   +---main
¦   ¦   +---java
¦   ¦       +---chuidiang
¦   ¦           +---ejemplos
¦   +---test
¦       +---java
¦           +---chuidiang
¦               +---ejemplos
+---target
    +---classes
        +---chuidiang
            +---ejemplos   //Aqui van los ficheros .class
```

Uso desde línea de comando

Continuación del resumen…

Ahora llega el momento duro. Debemos empezar a escribir el código, tanto de nuestro proyecto como de las clases de test de Junit

- Generar Jar
  - **mvn package** Esto primero compilará si es necesario, pasará las clases de test de JUnit y si no hay fallos, creará un archivo jar en el directorio target **EjemploMaven-1.0-SNAPSHOT.jar**

- Repositorios Maven

  Una de las grandes ventajas de maven son los repositorios (almacenes) de ficheros jar que se crea.

  Si miras en http://www.ibiblio.org/maven2/ tienes el repositorio oficial de jars de maven. Ahí están los groupId de casi todos los jar de libre distribución que puedas encontrar en internet. Tienes el log4j, commons-logging, JFreeChart, mysql-connector, etc, etc. Maven es capaz de bajarse cualquiera de estos jar si tu proyecto lo necesita.

  Todo lo que se baje maven de internet lo mete en un repositorio (almacen) local en tu pc, de forma que si lo necesita una segunda vez, no necesita descargárselo nuevamente de internet. Este directorio, habitualmente está en:

  - **$HOME/.m2 en unix/linux**
  - **C:\Documents and Settings\usuario\.m2 en windows**

Uso desde línea de comando

Continuación del resumen…

- Dependencias de nuestro proyecto. Una vez que sabemos que hay un montón de jars por el mundo a nuestra disposición, sólo tenemos que saber cómo hacer que maven se los baje cuando nosotros queramos. Para ello hay que editar el archivo POM

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>chuidiang.ejemplos</groupId>
  <artifactId>EjemploMaven</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Etc, etc...

Integración con las IDES

Se pueden crear proyectos o aplicaciones utilizando Maven desde las ides. Y también compartir proyectos entre todas. Esto permite utilizar lo mejor de cada una en cada proyecto.

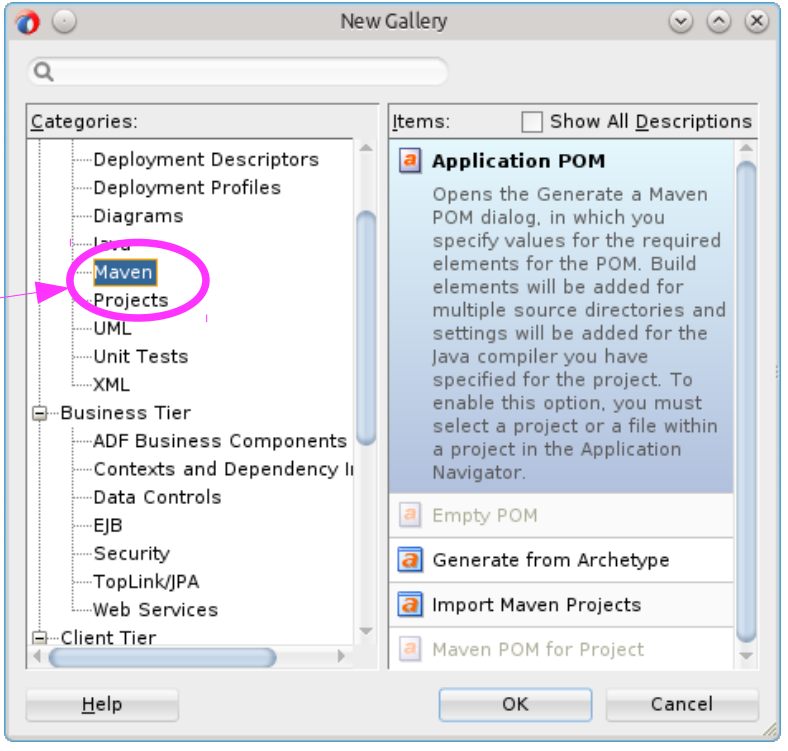
Creación de un proyecto con Maven: primera forma

Supongamos tener una aplicación creada, con sus proyectos dentro, pero sin MAVEN

Desde la aplicación a la cual queremos comenzar a administrar con Maven, hacemos:

File → New → Proyect

Y elegimos Maven



Creación de un proyecto con Maven: primera forma

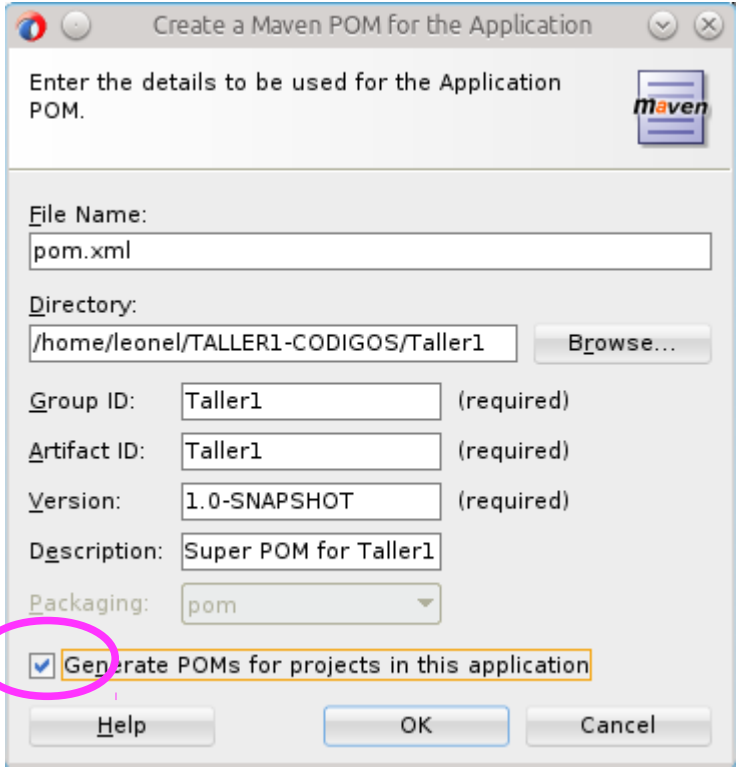
Desde la aplicación a la cual queremos comenzar a administrar con Maven, hacemos:

File → New → Proyect

Y elegimos Maven

Y luego tildamos la opción Generate POMs for projects in this application

Luego presionamos OK y esperamos.



Creación de un proyecto con Maven: primera forma

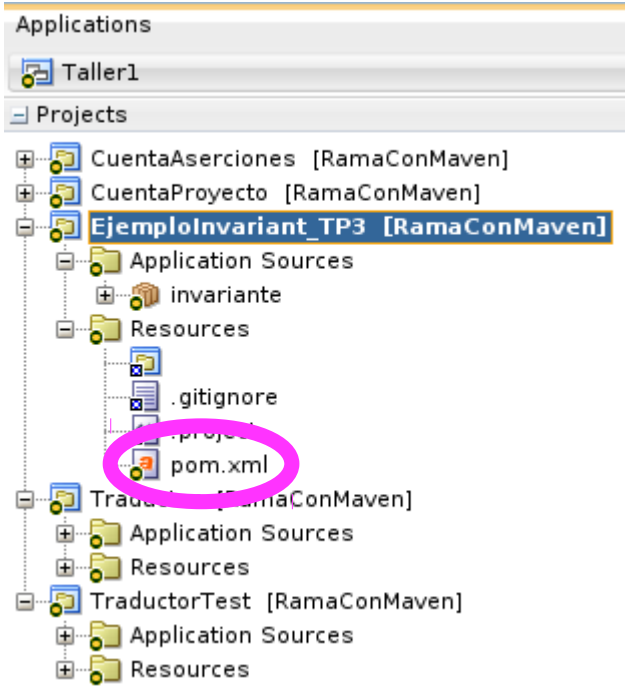
Nuestra aplicación se verá de la siguiente forma, se visualiza un archivo **pom.xml**

Este proyecto presenta también archivos propios del control de versiones con GIT y GITHUB.

En particular, a la rama local principal (master) le hice una nueva rama a la cual luego le apliqué la gestión con MAVEN.

De esta forma ya está la aplicación con Maven.

Ahora podría importar este proyecto desde ECLIPSE

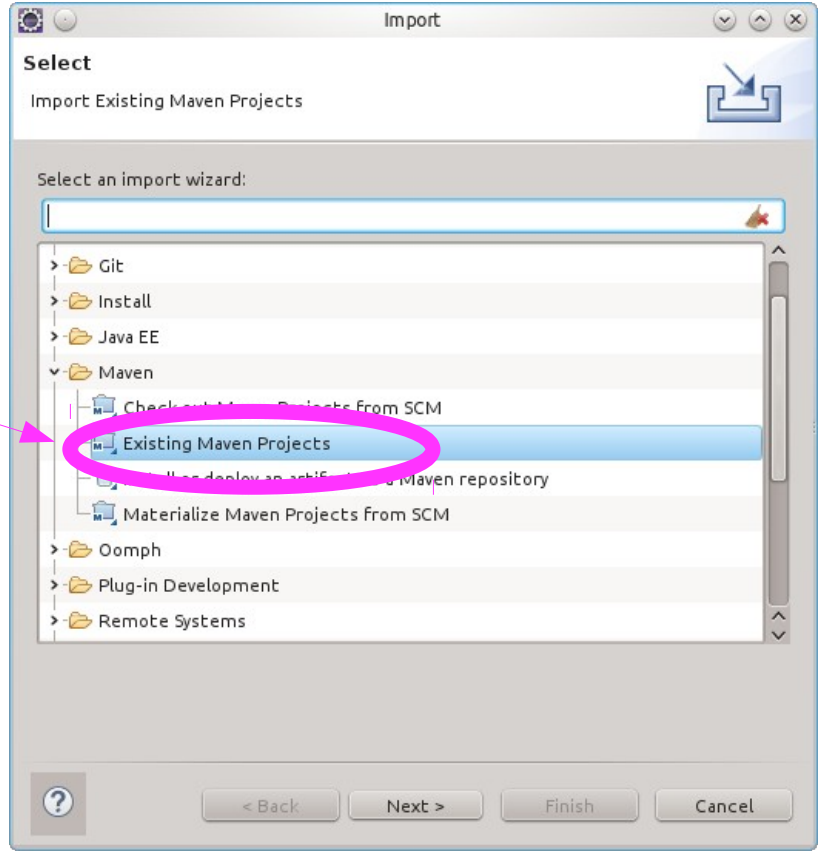


Importación desde Eclipse

Desde Eclipse, hacer **File → Import..**

Elegir la opción:

Maven → Existing Maven Projects



Importación desde Eclipse

Desde Eclipse, hacer **File → Import..**

Elegir la opción:

Maven → Existing Maven Projects

Con Browse...ubicar la carpeta en donde se encuentra el proyecto o aplicación a importar. Se puede importar un solo proyecto o toda la aplicación.

Avanzar en el wizard