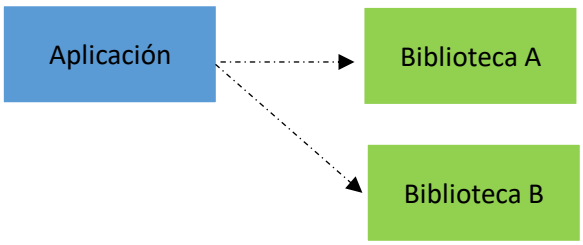
MAVEN

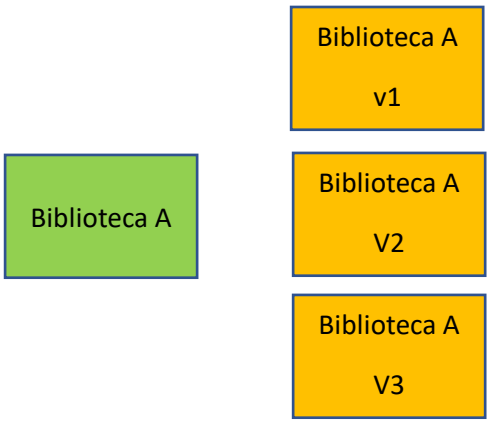
Dentro del mundo del desarrollo de aplicaciones Java (aunque los conceptos que se compartirán pueden extenderse a otras tecnologías), los arquitectos de software más pronto que tarde empiezan a utilizar frameworks tales como Spring, Hibernate, Apache CFX, JSF, etc; simplemente por el hecho de que aceleran el tiempo de desarrollo y facilitan el mantenimiento. Una vez que han aprendido a utilizar los frameworks, los desarrolladores van abordando proyectos de mayor complejidad. Y entonces en algún momento terminan encontrándose con Maven.

¿Qué es Maven?

Se hace necesaria una introducción antes de brindar un concepto o una definición de lo que es Maven. Volviendo al desarrollo de aplicaciones Java, ya sea en entornos stand-alone (JSE) como aplicaciones empresariales (JEE) es común empezar a utilizar bibliotecas que brindan funciones específicas. Estas bibliotecas brindan un conjunto de funcionalidades listas para ser utilizadas y tienen una ventaja fundamental; ya están validadas; por tanto, aquí también se cumple la premisa de que más pronto que tarde, el desarrollador usará bibliotecas disponibles en el lenguaje, algo representado en el siguiente esquema

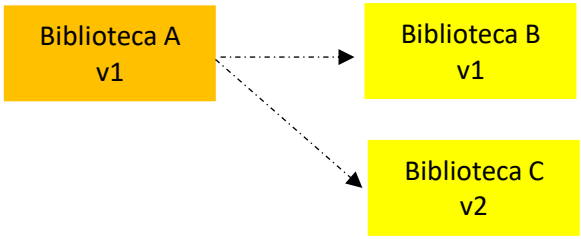


La gestión de bibliotecas es un aspecto de gran interés debido a que un mal uso puede generar diversos conflictos de compatibilidad, además mantener un repositorio ordenado de bibliotecas también es algo crucial. Por ese motivo las bibliotecas poseen una versión.



El versionamiento se vuelve algo fundamental, pero es más complejo de lo que parece. Es altamente probable que ciertas bibliotecas a su vez necesiten de otras para funcionar de manera correcta. De hecho, es lo esperado, por lo tanto, es evidente que se necesita más información para poder gestionar de manera correcta este repositorio de bibliotecas. Ud puede deducir que la complejidad de la gestión de la interrelación de las bibliotecas puede llegar a ser muy grande,

especialmente en plataformas de desarrollo de uso masivo, como lo es la plataforma Java, donde existen millones de bibliotecas para servicios muy diversos. Este concepto en el cual una aplicación/biblioteca requiere de una versión concreta de otra biblioteca se conoce con el nombre de dependencia. Esquemáticamente, esto está representado de la siguiente manera:



Aquí es donde entra Maven, que sin definir aún que es, podemos indicar que, para resolver el problema de las dependencias, introduce un concepto muy interesante: **el artefacto**. En Maven un artefacto, es una entidad que además de contener las clases desarrolladas, contiene información sobre su versión, dependencias, grupo, etc:

Este conjunto de clases con su respectiva información de versionado es lo que permite que un artefacto Maven sea un bloque de código reutilizable.

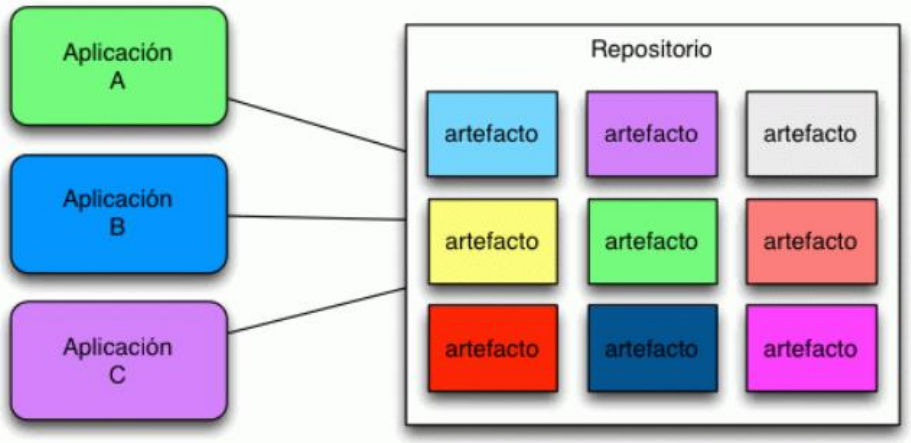
Hay que realizar algunas aclaraciones, paradójicamente para aquellos que poseen mayor experiencia en el desarrollo con Java:

- Un artefacto Maven no es lo mismo que un archivo o biblioteca jar (archivo java). Un archivo jar es un bloque de código reutilizable previamente compilado (contiene los .class o bytecodes) mientras que un artefacto Maven no contiene bloques de código previamente compilado.
- Un artefacto Maven no es lo mismo que un bloque de código fuente java (los .java), debido a que estas no contienen ninguna información de las dependencias que posea.

¿Por qué se solicita que un artefacto brinde la información indicada (nombre, grupo, versión y dependencias)?

Una vez que Ud define correctamente todos los artefactos que necesita, la idea es alojarlos en un repositorio central, para poder mantenerlos y distribuirlos. Es decir, existe en la nube (internet) un repositorio donde existen artefactos que pueden ser reutilizados por otros

desarrolladores para crear su aplicación, esto es, a partir de los artefactos alojados en diversos repositorios centrales, se crean aplicaciones formadas básicamente por artefactos:



Entonces, Ud podrá entender que es necesario identificar unívocamente cada artefacto para que no existan en el mismo repositorio (de nivel mundial, como podrá apreciar) dos artefactos iguales. Si partimos de la idea de que básicamente todo artefacto cumple con ciertos principios muy elementales:

- Posee un nombre o un identificador
- Pertenece a una organización o persona
- Posee una versión concreta

Es entendible que para definir un artefacto se deba proveer las siguientes partes (además del bloque de código):

- Grupo: representa realmente un identificador a nivel global de uno o más artefactos, permite establecer un nombre único a nivel global en el que se organizan un conjunto de artefactos, pertenecientes a una organización o persona concreta, los cuales se ponen a disposición de otros desarrolladores. ¿Cómo definir este nombre? Existe una regla muy interesante. No existen a nivel mundial URL de páginas web repetidas. Por este motivo se recomienda formar un identificador basado en la jerarquía de paquetes de la aplicación a desarrollar que se corresponda con las partes de una URL invertida. Por ejemplo, la URL de la unidad académica a la que pertenece esta carrera es www.fi.unju.edu.ar. Debido que la generalidad está en el extremo derecho y la especificación del lado izquierdo (fi, que representa a la unidad académica Facultad de Ingeniería está ubicada a la izquierda de unju [que representa a la Universidad Nacional de Jujuy]. La facultad pertenece a la universidad, por eso esta última está a la derecha), para la generación de este identificador o grupo se invierte el mismo, es decir:

  ar.edu.unju.fi.componente

  Donde **componente** expresa normalmente el nombre del artefacto y coincide con el nombre de la aplicación.
- Nombre: Será el nombre del componente.
- Versión: permite indicar la versión del artefacto.
- Las dependencias hacen referencias al grupo, nombre y versión de otros artefactos que el presente artefacto requiere para funcionar de manera adecuada.

Gracias a este esquema, un artefacto puede rastrear en el repositorio de sus dependencias y descargarlas a la aplicación con la certeza y confianza de que funcionará adecuadamente al momento de generar y ejecutar la aplicación.

Maven es una implementación de las ideas vertidas hasta este momento (y mucho más, pero nos centraremos solo en el aspecto de la gestión de dependencias):

Es un gestor de dependencias de un proyecto elaborado con Maven (comúnmente denominado proyecto Maven). Se encarga de ubicar y descargar las dependencias del proyecto, y luego las añade al classpath del proyecto.

Para lograr esto el proyecto Maven posee un archivo denominado pom.xml (de Project Object Model), el cual es el archivo fundamental donde se establece la configuración. Este archivo cumple diversas funciones: describe el proyecto, los plugins que utiliza, informaciones sobre el sistema de control de versiones, el enlace a repositorios externos al repositorio central de Maven (pueden existir dependencias que no estén en el repositorio de Maven), y como puede imaginar, es en este archivo donde se definen las dependencias del proyecto. Todo proyecto Maven posee al menos un archivo pom.xml. Más adelante estudiaremos con mayor detalle este archivo; sin embargo, observe el siguiente ejemplo de definición de un archivo POM para un proyecto java:

```
  https://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation)
1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://m
4   <modelVersion>4.0.0</modelVersion>
5   <groupId>ar.edu.unju.fi.firstporject</groupId>
6   <artifactId>firstproject</artifactId>
7   <version>0.0.1-SNAPSHOT</version>
8 </project>
```

Al ser un archivo xml, se utilizan tags para determinar el significado de diversas informaciones.

El tag <project> es utilizado para establecer atributos que indiquen que es tipo de proyecto Maven, la ubicación tanto del repositorio como la de la versión del gestor de dependencias.

El tag <movelVersion> permite especificar explícitamente la versión de Maven.

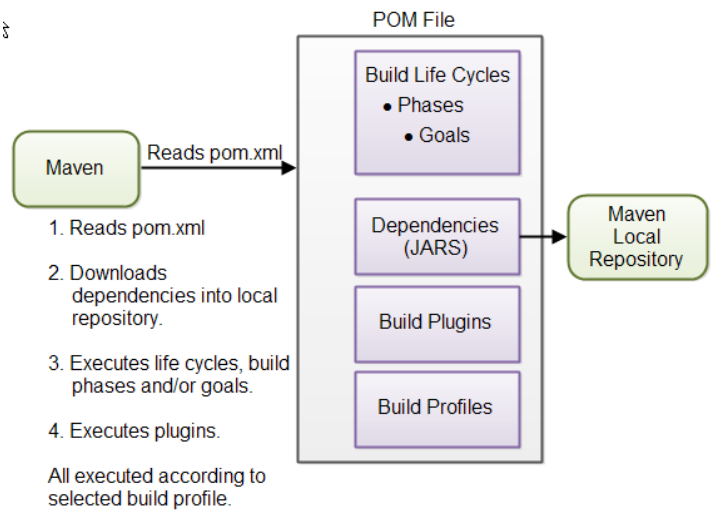
El tag <groupid> permite establecer el grupo del artefacto. Observe que se refiere al dominio de `ar.edu.unju.fi`, por lo tanto, hace referencia al conjunto de artefactos disponibles de este espacio de soluciones. El tag <artifactid> permite definir el nombre concreto del artefacto. Como se indicó previamente hac referencia al nombre del proyecto. Para algunos proyectos más relevantes, tales como spring; permiten indicar las familias de funcionalidades, aunque sobre esto se profundizará más adelante. El tag <version> permite indicar la versión concreta del artefacto. Si ud no lo suministra, Maven asumirá que hace referencia a la última versión disponible en el repositorio. Estas etiquetas son lo mínimo y obligatorio que se debe suministrar al momento de declarar una dependencia.

Si bien en el ejemplo anterior no están presentes, haremos referencia a algunas de las etiquetas más importantes. Un tag importante es <scope>. Esta etiqueta permite indicar el alcance de nuestra dependencia y su transitividad. Existen 6 tipos de valores disponibles para esta etiqueta:

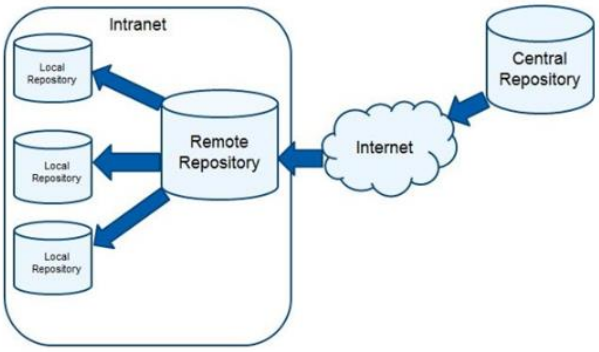
- compile: es la opción por defecto sino se especifica la etiqueta scope. Indica que es necesario compilar la dependencia. La dependencia además se propaga en los proyectos dependientes.
- provided: Es como la anterior, pero asume que el contenedor es quien posee la biblioteca. Un claro ejemplo es cuando desplegamos en un servidor de aplicaciones el producto creado; es bien sabido que el servidor de aplicaciones posee por defecto un conjunto bastante amplio de bibliotecas, que seguramente son utilizadas en el proyecto, así que por lo tanto no es necesario desplegar la dependencia.
- runtime: La dependencia es necesaria en tiempo de ejecución, pero no es necesaria para compilar.
- 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>. Es el caso en el cual el desarrollador crea su propia biblioteca y la usa regularmente en sus proyectos. Para incluir esta biblioteca en un proyecto Maven se utiliza esta opción.
- import: este solo se usa en la sección dependencyManagement; y tiene que ver con la transitividad de dependencias. No se profundizará en esta opción.

El proceso de búsqueda de dependencias en proyectos Maven

Las dependencias de los proyectos que el desarrollador crea se almacenan en una carpeta local denominada .m2. Cuando se ejecuta un proyecto Maven (o cuando se desarrolla y prueba el mismo) se consulta el archivo pom.xml para determinar las dependencias del proyecto y se verifica en primera instancia, si estas existen en la carpeta .m2. (el repositorio remoto). En caso de que no existan allí, Maven consultará su repositorio central y descargará las dependencias necesarias al repositorio remoto (.m2) para que puedan ser utilizadas por todos los proyectos locales. Este aspecto es interesante, por cuanto Maven considera los proyectos Maven creados como artefactos de un repositorio local.



En la próxima compilación, el proyecto ya dispondrá de las dependencias dentro del repositorio remoto. Por tanto, si se despliega o ejecuta el proyecto, este inspeccionará en su repositorio local, y en caso de no encontrar las dependencias indicadas las buscará en el repositorio central donde si las encontrará. De modo que, no será necesario descargarla otra vez del Maven Central Repository (MCR), ya que esas dependencias se pueden satisfacer sin necesidad de contactarse al MCR por medio de internet.



El repositorio .m2 (repositorio remoto) por defecto está ubicado por defecto en directorio:

`${user.home}/.m2/repository`

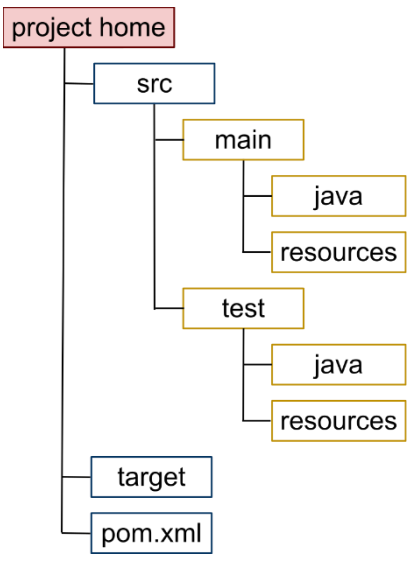
- Unix/Mac OS X – `~/.m2/repository`
- Windows – `C:\Users\{your-username}\.m2\repository`

Y evidentemente las dependencias, físicamente son los archivos jar´s organizados en carpetas, que se corresponden con el groupid indicado en los proyectos.

Estructura de un proyecto Maven

Ante todo, Apache Maven es un software para gestionar proyectos. Al crear un proyecto Maven, automáticamente se generará una estructura de carpetas muy concreta ya que viene predefinida.

Cada uno de los directorios, tiene una funcionalidad dentro del proyecto:



- src/main/java: contiene el código fuente; es decir aquí el desarrollador crea o edita las clases con extensión .java. El contenido en este directorio se conoce con el nombre de módulo.
- src/main/resources: contiene los recursos estáticos (XML, propiedades, imágenes, etc) que necesita el módulo para funcionar correctamente.
- src/test/java: contiene los archivos para realizar el testing del código, es decir las clases sometidas a la verificación del funcionamiento del módulo.
- Src/test/resources: almacena los archivos que genera Maven al usar los comandos de la herramienta (compile, package, etc. Estos comandos quedan fuera del estudio directo; ya que los aplicaremos dentro del IDE).
- pom.xml: el Project Object Model (POM) es el encargado de gestionar y construir los proyectos, contiene el listado de dependencias que son necesarias para que el proyecto funciones. También contiene una definición de plugins, etc. Toda la información del proyecto está basada en este archivo. Tiene extensión .xml y desde la propia página web oficial de Apache Maven, lo definen como el núcleo central del proyecto.

- superpom.xml: Todo proyecto Maven hereda del "superpom" que es un pom.xml padre, donde se define el build por defecto, para que todos los proyectos usen la estructura definida en él.

Las imágenes utilizadas corresponden a sitios indicados en la bibliografía

Bibliografía

https://www.arquiecturajava.com/maven-tutorial/

Documentación oficial de Maven: https://maven.apache.org/