Contenido

Gestores de dependencias. Maven. 2
¿Qué son los gestores de dependencias? 2
Problemas de las dependencias 2
Soluciones 2
Gestores de dependencias en Java 3
Maven 3
Gradle 3
Ampliando en Maven 5
Convención sobre configuración 5
Reutilización 5
Ciclo de vida 6
Como Resumen 6
Creación de un proyecto en Maven 7
El archivo POM 13
Ejecución de un archivo Maven 16
Para profundizar… 23

Gestores de dependencias. Maven.

Por Francisco García

¿Has tenido problemas cuando tienes que incorporar librerías en tu aplicación? ¿Es un follón mantener las versiones de las mismas? Los gestores de dependencias son tu solución, y en especial Maven.

¿Qué son los gestores de dependencias?

En el campo del software una **dependencia** es una aplicación o una biblioteca requerida por otro programa para poder funcionar correctamente. Por ello se dice que dicho programa depende de tal aplicación o biblioteca.

Problemas de las dependencias

Resolución de dependencias durante la instalación

Algunas formas de instalación de software, como el uso de instaladores o la compilación del código fuente, no buscan ni descargan automáticamente las dependencias del programa. Lo que se hace en estos casos es indicar al usuario las aplicaciones o bibliotecas necesarias para que éste las busque, descargue e instale manualmente.

Falta de dependencias tras una desinstalación

En ciertas ocasiones, cuando se elimina software que hace uso de alguna biblioteca compartida, ésta también se desinstala con él, ocasionando el mal funcionamiento del sistema operativo o de una aplicación que hacía uso de ella.

Problemas de versiones

En algunos casos, diferentes programas hacen uso de la misma biblioteca, pero necesitan versiones distintas. Por ello han de mantenerse en el sistema operativo diferentes versiones de una misma biblioteca que, en algunos casos, interfieren entre sí, provocando el mal funcionamiento de los programas.

Soluciones

Los problemas de dependencias de software se pueden resolver de distintas maneras. Una de ellas es proporcionar un instalador que incluya todas las dependencias o las descargue automáticamente. Otro método alternativo es el de los gestores de paquetes, que calculan las dependencias cada vez que se instala nuevo software y las descarga. La desventaja más notoria de estos sistemas es probablemente que puede haber conflictos con bibliotecas compartidas de diferente versión. Sin embargo, en el caso de los gestores de paquetes es menos probable que esto ocurra, ya que la mayoría de los paquetes usados por los gestores de paquetes suelen ser creados por la misma persona o el mismo equipo de personas, con lo que pueden resolver dichos conflictos.

Gestores de dependencias en Java

Ya desde hace unos años todo el mundo utiliza un gestor para construir un proyecto y obtener sus dependencias. Ahora ya no podemos vivir sin estas tecnologías, pero antes se tenían que importar todas las dependencias a mano y meter a una persona en un proyecto se podía convertir en un infierno hasta que tenía todo el entorno configurado para empezar a trabajar. Hoy por hoy, entrar en un proyecto debería ser tan fácil como descargarlo y compilarlo para tener todo funcionando sin tener que perder mucho tiempo en su configuración.

Hay dos gestores que se han convertido en los más utilizados: Maven y Gradle.

Maven

Maven es una herramienta de software libre para gestionar proyectos Java lanzada en 2002. Se basa en ficheros xml y viene con objetivos definidos para realizar ciertas tareas como la compilación del código y la generación del proyecto empaquetado, pero Maven también se encarga de obtener todas las dependencias del proyecto como de subirlo al repositorio final para que otros proyectos que dependan de él puedan usarlo.

Además, Maven se puede usar con multitud de añadidos aportados por la comunidad de desarrolladores, por ejemplo, hay plugins que se pueden usar para funciones como desplegar el proyecto en un servidor, pasar el proyecto a un analizador de calidad de código o generar la cobertura de test que tiene la aplicación.

Gradle

Gradle también es una herramienta de software libre lanzada en 2007 pero sirve para gestionar proyectos Java, Groovy o Scala, aunque ya soporta otros lenguajes. Esta herramienta se configura con ficheros DSL basados en Groovy, en vez de los xml que usaba Maven.

Gradle ha surgido para solucionar el problema de la gestión de la construcción, dependencias y despliegue de los proyectos que tengan otro lenguaje diferente a Java, aunque este también

soporta Java. Día a día va creciendo el uso de esta tecnología y al igual que Maven, tiene una gran cantidad de plugins que se pueden añadir para multitud de tareas relacionadas con la construcción, la gestión de dependencias y el despliegue.

Ampliando en Maven

Maven es una herramienta de software para la gestión y construcción de proyectos Java creada por Jason van Zyl, de Sonatype, en 2002. Es similar en funcionalidad a Apache Ant (y en menor medida a PEAR de PHP y CPAN de Perl), pero tiene un modelo de configuración de construcción más simple, basado en un formato XML. Estuvo integrado inicialmente dentro del proyecto Jakarta pero ahora ya es un proyecto de nivel superior de la Apache Software Foundation.

Maven utiliza un Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.

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. Actualmente existe un plugin para .Net Framework y es mantenido, y un plugin nativo para C/C++ fue alguna vez mantenido por Maven 1.

Convención sobre configuración

La filosofía general de Maven es la estandarización de las construcciones generadas por seguir el principio de Convención sobre Configuración, a fin de utilizar modelos existentes en la producción de software. Esta estrategia necesariamente restringe ampliamente la variabilidad, lo que se refleja en la exhortación de Maven a adherirse a su modelo de proyecto. Mientras que Maven es adecuado para nuevos proyectos, los proyectos complejos ya existentes pueden ser simplemente no adaptables para que utilicen Maven. La falta de restricciones de la convención de capas del proyecto fue hecha de alguna manera más configurable con el lanzamiento de Maven 2.

Reutilización

Maven está construido alrededor de la idea de reutilización, y más específicamente, a la reutilización de la lógica de construcción. Como los proyectos generalmente se construyen en patrones similares, una elección lógica podría ser reutilizar los procesos de construcción. La principal idea es no reutilizar el código o funcionalidad (como Apache Ant), sino simplemente cambiar la configuración o también código escrito. Esa es la principal diferencia entre Apache Ant y Apache Maven: el primero es una librería de utilidades y funciones buenas y útiles, mientras que la otra es un framework configurable y altamente extensible.[4]

Aunque Maven es configurable, históricamente el proyecto Maven ha enfatizado seriamente que los usuarios deben adherirse a su concepto de un modelo de proyecto estándar tanto como sea posible.

Ciclo de vida

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

1. *compile*: Genera los ficheros .class compilando los fuentes .java
2. *test*: Ejecuta los test automáticos de JUnit existentes, abortando el proceso si alguno de ellos falla.
3. *package*: Genera el fichero .jar con los .class compilados
4. *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.
5. *deploy*: Copia el fichero .jar a un servidor remoto, poniéndolo disponible para cualquier proyecto Maven con acceso a ese servidor remoto.

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 algunas metas que están fuera del ciclo de vida que pueden ser llamadas, pero Maven asume que estas metas no son parte del ciclo de vida por defecto (no tienen que ser siempre realizadas). Estas metas son:

1. *clean*: Elimina todos los .class y .jar generados. Después de este comando se puede comenzar un compilado 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 que 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.

Pero estas metas pueden ser añadidas al ciclo de vida a través del Project Object Model (POM).

Como resumen:

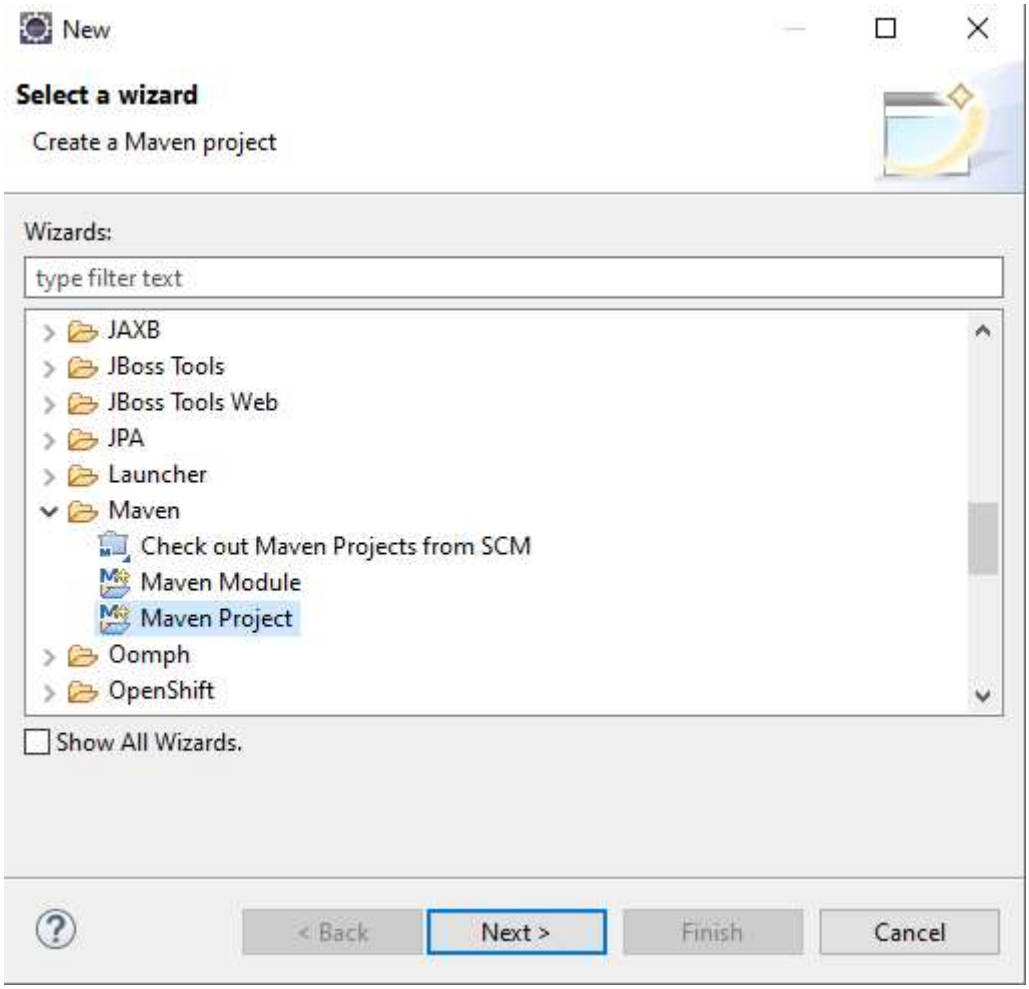
● Maven es una herramienta de código abierto con el objetivo de simplificar los pasos de compilación y generación de ejecutables a partir del código fuente.

● Antes si queríamos compilar y generar ejecutables de un proyecto teníamos que ir analizando el código para ver qué teníamos que compilar, qué librerías se estaban utilizando y dónde se debían insertar, así como las dependencias que tuviera nuestro proyecto a la hora de compilar. Por tanto, podríamos decir que Maven es una herramienta capaz de gestionar un proyecto sofware completo.

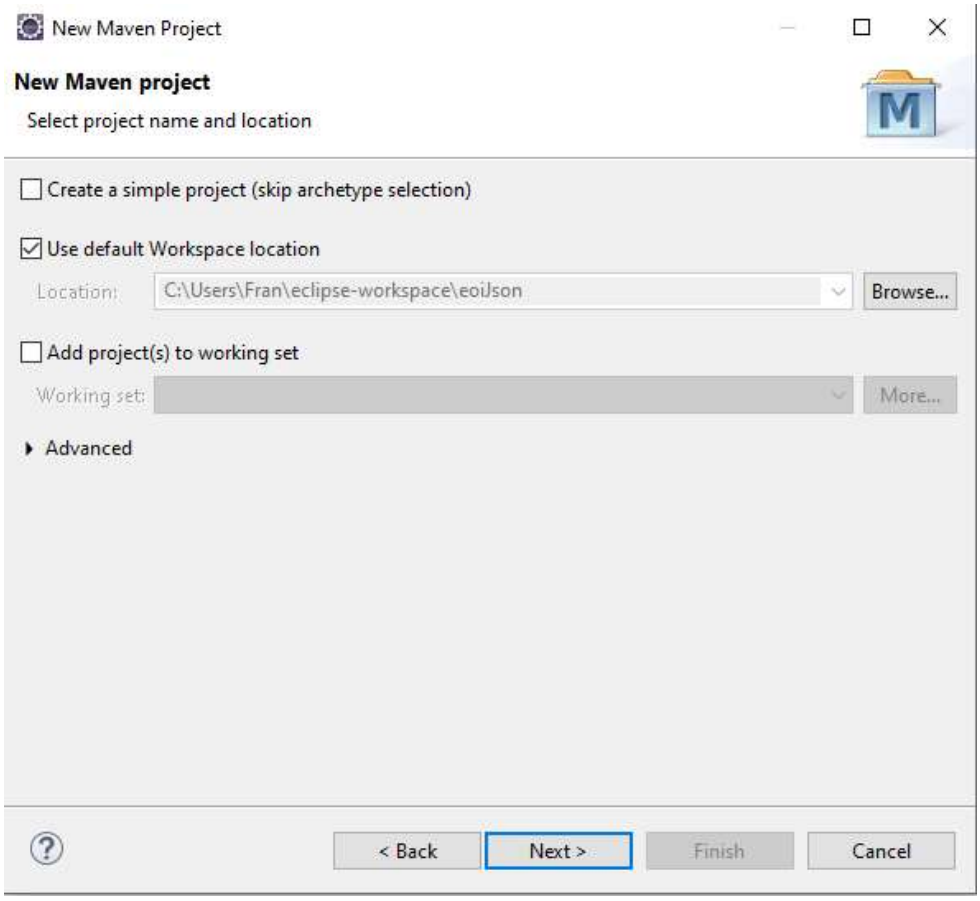
● Para el caso de la gestión de dependencias entre módulos y versiones de librerías, Maven nos ayuda a la hora de buscarlas y descargarlas, de forma que nosotros nos olvidamos por completo de esa labor y todo eso gracias al fichero de configuración de Maven llamado POM.

Creación de un proyecto en Maven

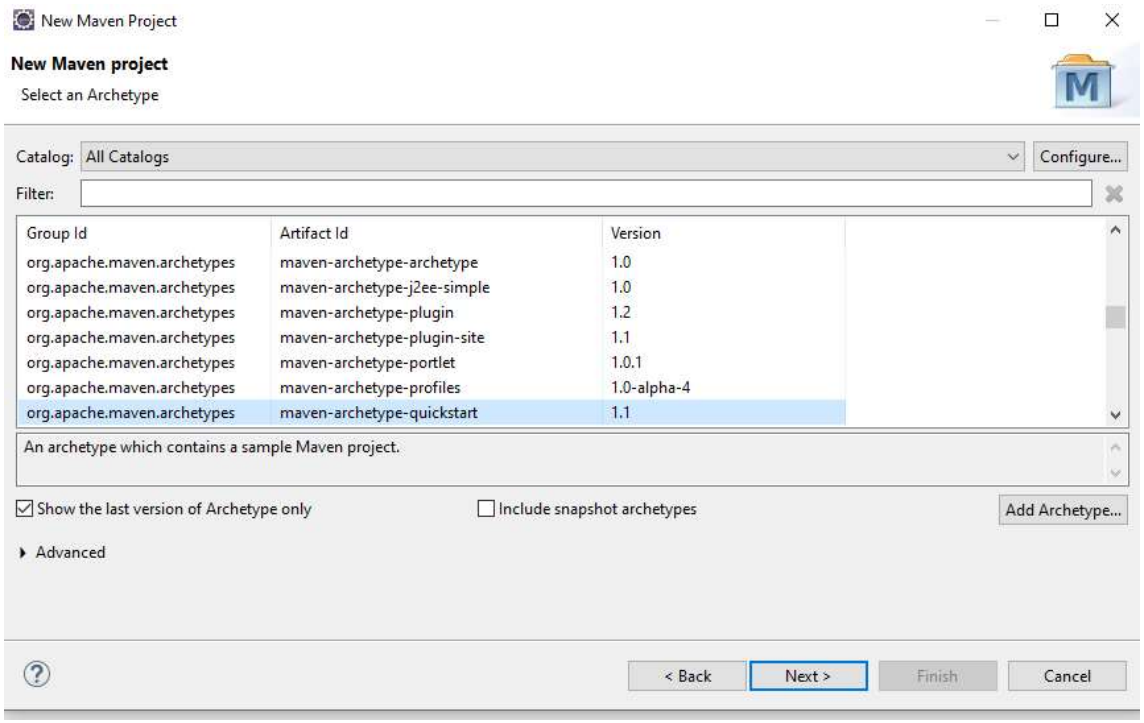
Teniendo la última versión de Eclipse instalada, para crear un proyecto Maven, accedemos a File-> New ->Other -> Maven->Maven Project:



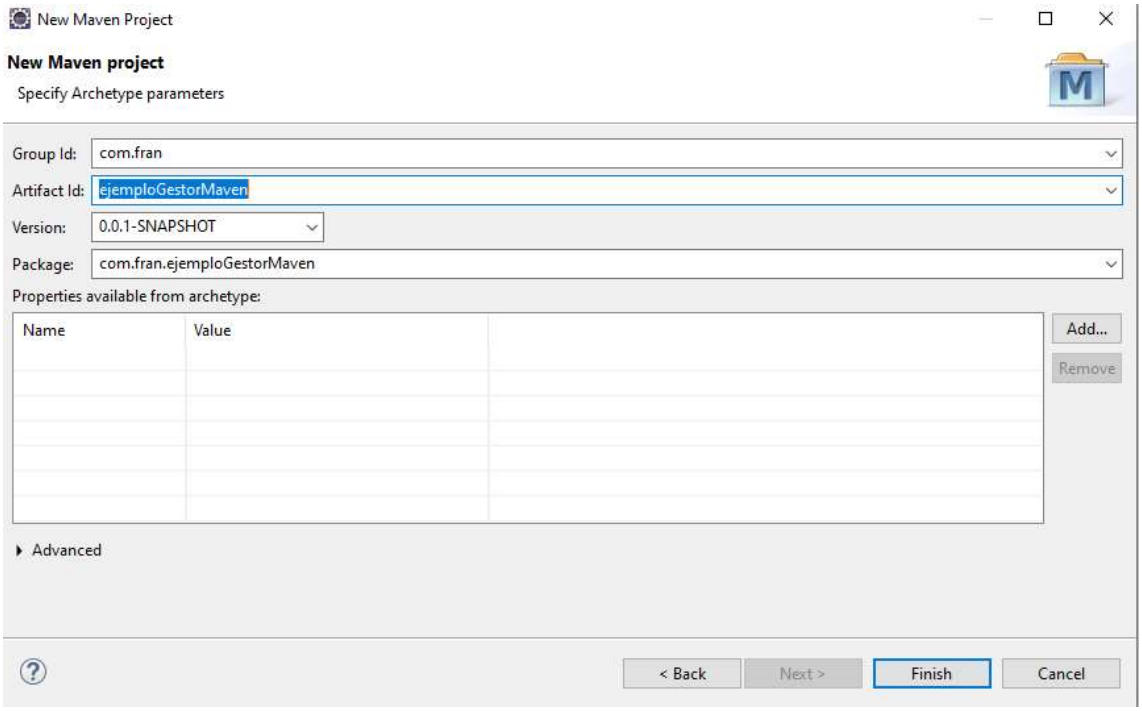
Pulsamos Next y accedemos a la siguiente ventana:



Dejaremos el WorkSpace por defecto y continuamos:



Seleccionamos el maven-archetype-quickstart (yo tengo la versión 1.1 pero quizá cuando se lean estas líneas vaya por una versión posterior).



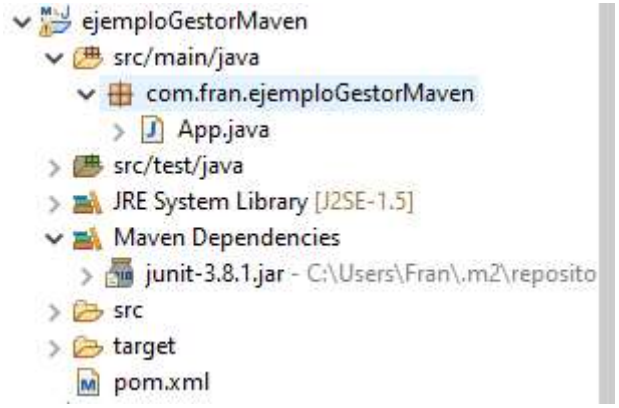
– Group Id: El nombre de nuestra compañía .

– Artifact Id: Coincide con el nombre de nuestro proyecto.

– Version: La versión del proyecto, por defecto es la 1.0

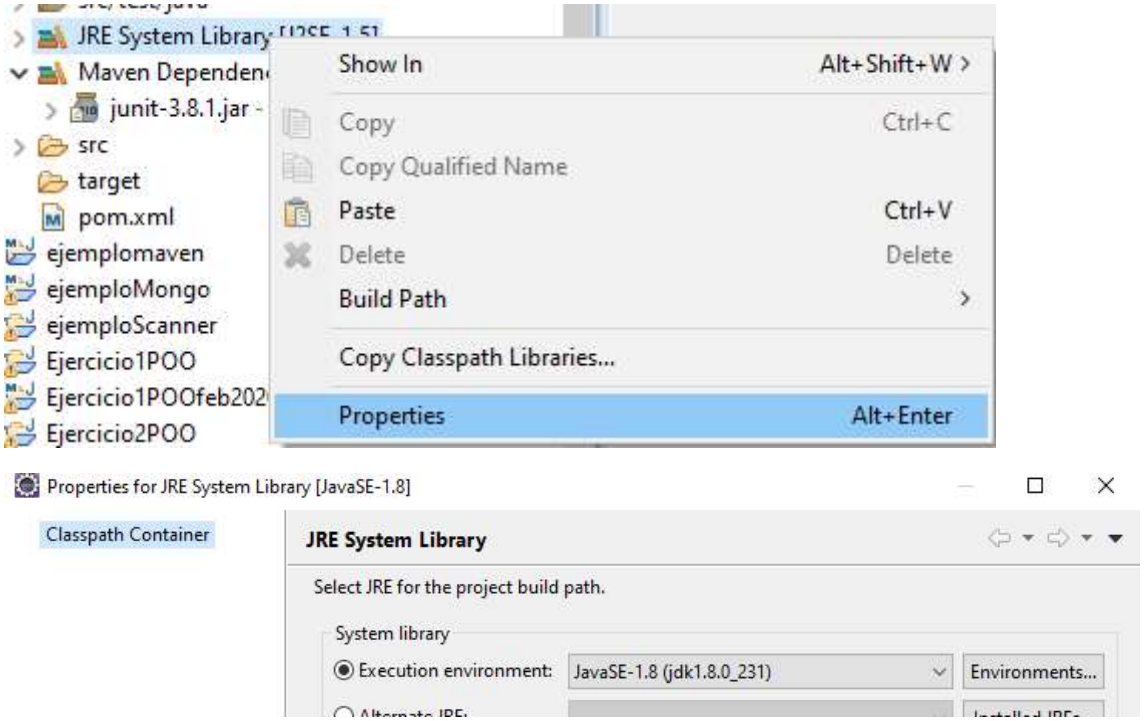
– Package: El nombre del paquete, este campo es opcional pero por defecto siempre nos aparecerá: group Id + project name

Nos habrá creado un proyecto como el siguiente, donde podemos observar algunas carpetas y archivos interesantes:



Dentro del paquete com.fran.ejemploGestorMaven encontramos la clase creada por defecto llamada App.java donde dentro tenemos un Main que ejecuta el típico “Hola Mundo”. Sería como la carpeta base de un típico proyecto en Java.

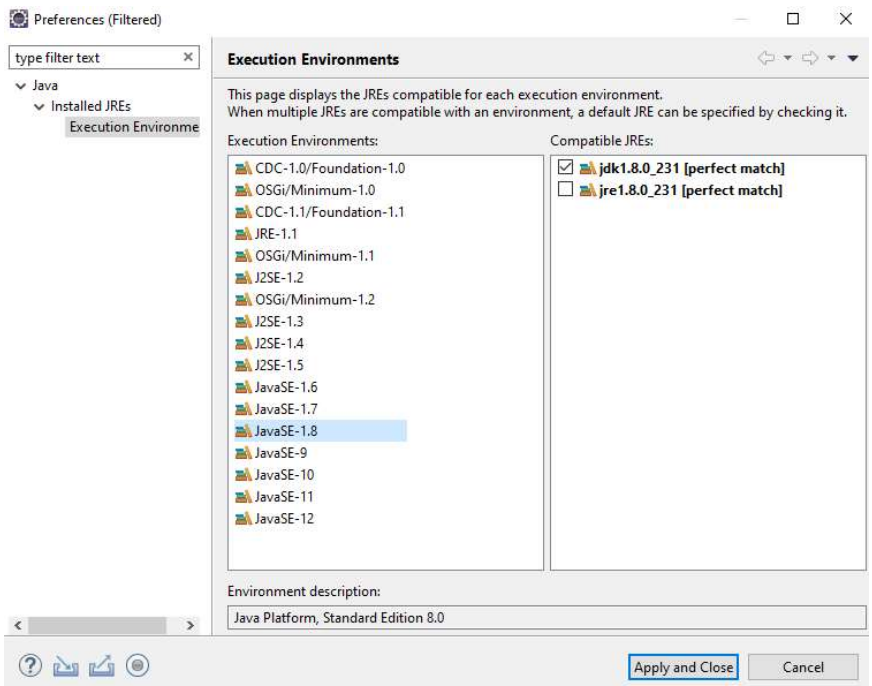
Podemos cambiar la versión de Java a la 1.8:



NOTA IMPORTANTE: Selecciona un JDK y no un JRE. Si no tienes ningún JDK puedes descargarlo de la página de Oracle:

https://www.oracle.com/java/technologies/javase-jdk8-downloads.html

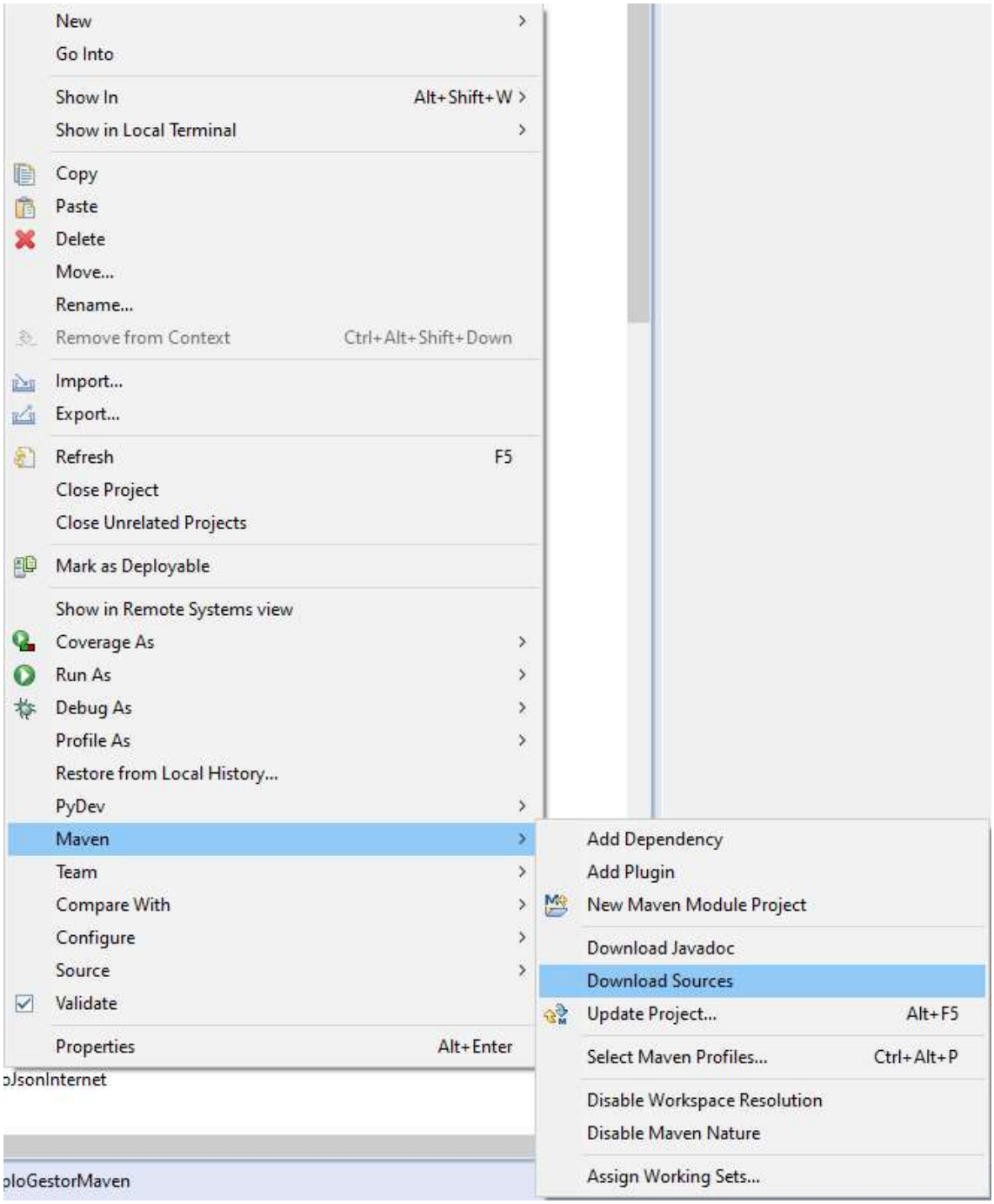
Si lo tenemos descargado y nos aparece e instalado y sin embargo aparece un JRE puedes cambiarlo pulsando Environments y seleccionándolo:



Podemos ver que la carpeta Maven Dependencies tiene una dependencia añadida en el proyecto por defecto, que es la de JUnit para poder realizar pruebas unitarias. Observamos que la ruta donde nos guarda las dependencias es a nivel de ordenador, en una carpeta de nuestro disco duro:

Esto es importante, ya que las dependencias no son propias de un proyecto, sino que se compartirán entre diferentes proyectos, lo que nos ahorrará bastante tamaño en los proyectos ya que no hay que insertar la librería por cada proyecto que estamos haciendo y solo la "linkará" cuando creemos el archivo jar ejecutable.

Por otro lado, también hay que tener en cuenta que al no estar insertadas las librerías a nivel de proyecto, si lo compartimos con alguien, este **deberá descargarse dichas librerías donde importe nuestro proyecto,** lo cual es sencillo, ya que en dicho caso al importar el proyecto solo tiene que dar botón derecho sobre el mismo y en la carpeta Maven, ejecutar 2 opciones: Download Sources y Update Project:



Con eso conseguirán descargar las dependencias si importamos un proyecto realizado por otro, o si estamos compartiendo un proyecto a través de Git.

El archivo POM

POM significa modelo de objeto de proyecto. Es una representación XML de un proyecto Maven contenido en un archivo llamado pom.xml. Un POM contiene archivos de configuración, así como los desarrolladores involucrados y los roles que desempeñan, el sistema de seguimiento de defectos, la organización y las licencias, la URL donde vive el proyecto, las dependencias del proyecto y todas las otras pequeñas piezas que entran en juego para dar código vida. Un POM es una ventanilla única para todo lo relacionado con él. De hecho, en el mundo de Maven, un proyecto no necesita contener ningún código, simplemente un pom.xml.

Por defecto, nuestro proyecto nos ha creado el siguiente archivo pom.xml:

ejemploGestorMaven/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.fran</groupId>
  <artifactId>ejemploGestorMaven</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>ejemploGestorMaven</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>
```

– modelVersion: por defecto debería estar establecido en la 4.0.0

– groupId: Nuestra compañía

– artifactId: el nombre de nuestro proyecto

– version: la versión del proyecto

– packaging: tipo de empaquetado de nuestro proyecto. Si hiciéramos nosotros desde cero este archivo y no lo ponemos no pasa nada ya que por defecto es jar

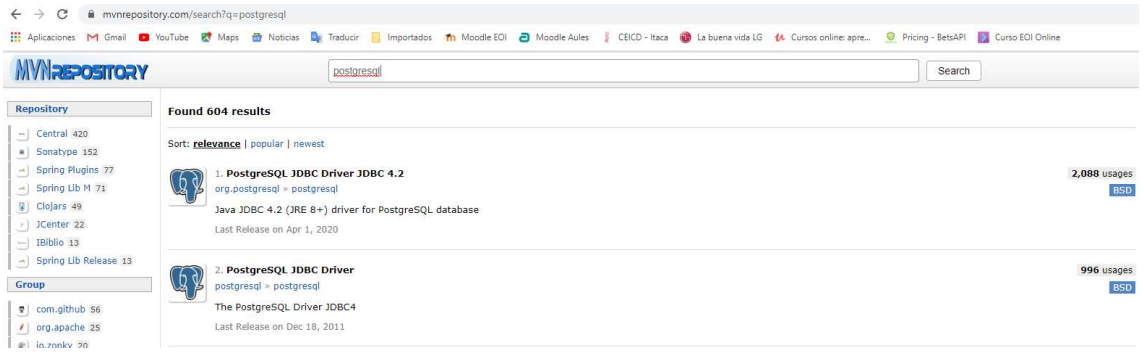
– La configuración dentro del properties corresponde a la codificación del proyecto (UTF-8)

Vemos que por defecto ya tenemos creada una dependencia (JUnit) dentro de la etiqueta dependencies. Pero si quiero añadir una nueva, ¿Cómo conocemos nosotros su groupId, artifactId y version?

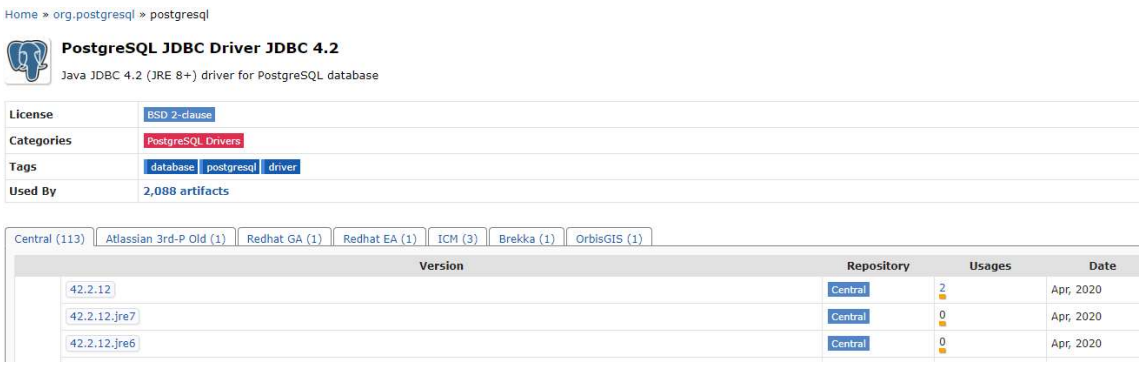
Para ello existe la siguiente página donde podemos encontrar las dependencias que busquemos:

https://mvnrepository.com/

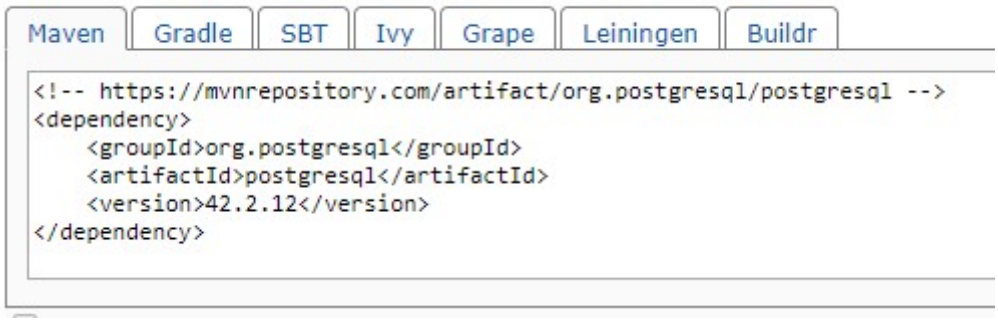
Para el ejemplo, vamos a suponer que queremos insertar la dependencia JDBC de PostgreSql para nuestro proyecto, lo buscamos:



Selecciono el que me interesa (en este caso el primer enlace) y que normalmente será el de más uso o el más moderno.



En este caso, yo pincharía el 42.2.12 ya que no es una versión que ponga en Beta y es reciente, luego copio la dependencia a insertar:



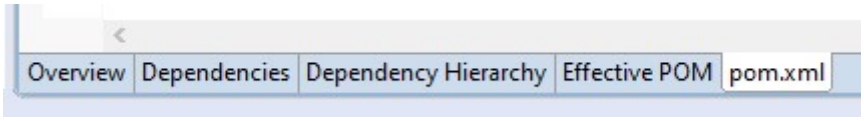
Y la insertaría en el pom.xml:

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

        <!-- https://mvnrepository.com/artifact/org.postgresql/postgresql -->
        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <version>42.2.12</version>
        </dependency>


    </dependencies>
</project>
```

(podéis darle a botón derecho, source->format para que os lo deje bien formateado)

Una vez que le deis Control+S para guardar, descargará en segundo plano la librería de postgresql y ya estará listo para ser usado en el programa. Podéis ver que ha sido descargado y se ha linkado en la carpeta Maven Dependencies.

A mí personalmente me gusta la vista del pom.xml en formato fichero xml, pero podéis ver que hay otras formas de ver el fichero cambiando en las pestañas de abajo:



Ejecución de un archivo Maven

Para ejecutar el contenido del proyecto “en local” podemos hacer como con cualquier proyecto Java, acceder a la clase que queremos probar y darle al botón de ejecución  :

ejemploGestorMaven/pom.xml | App.java

```
1  package com.fran.ejemploGestorMaven;
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
```

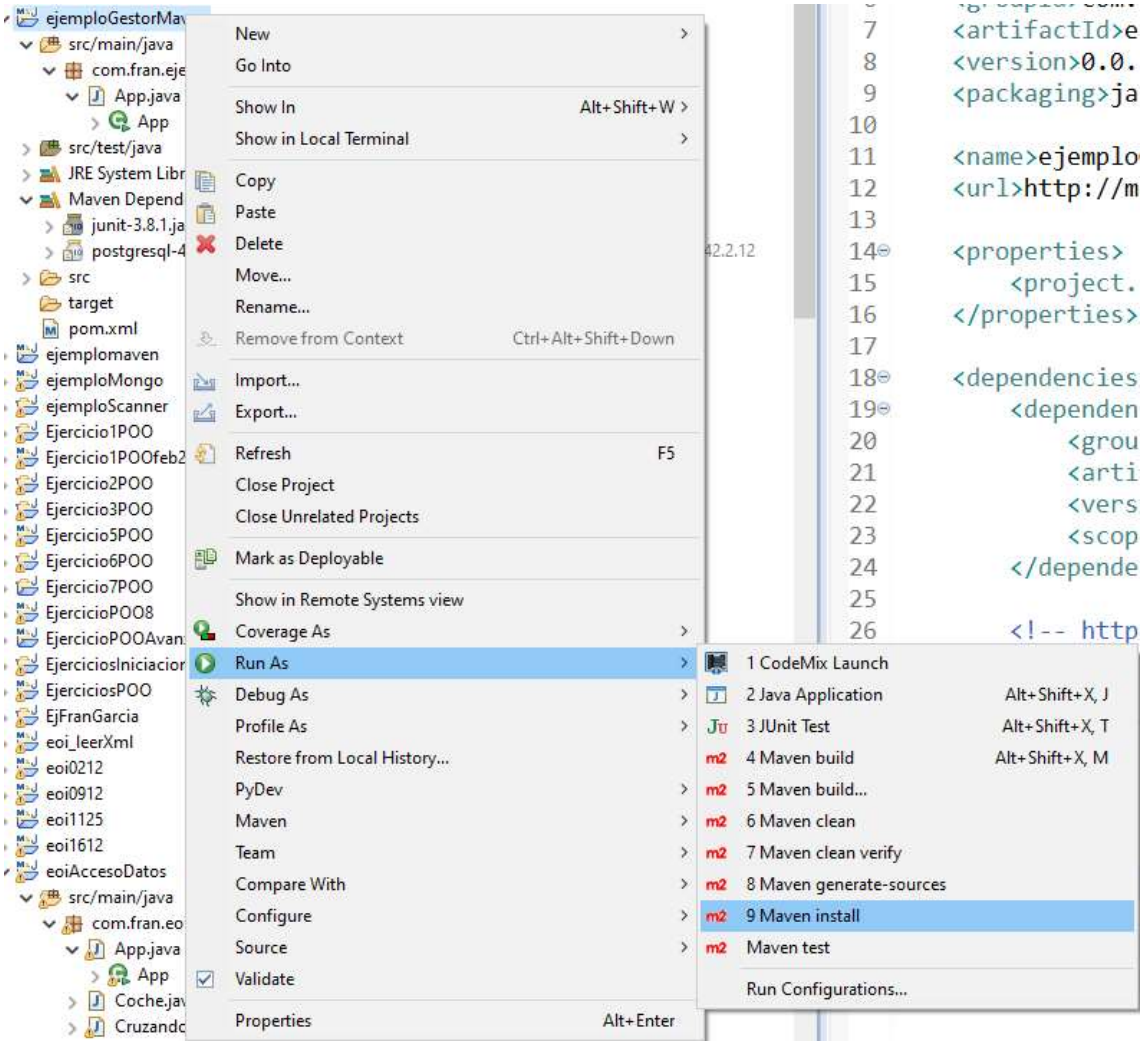
Veamos ahora como ejecutar ahora el proyecto Maven. Vamos a lanzar por primera vez nuestro proyecto **Maven** y aprovecharemos para ver los ciclos de vida posibles que tiene.

Hacemos clic con el botón derecho sobre nuestro proyecto y vamos a **Run As**. Ahí nos aparecerán diversas opciones:

¿Qué hace cada una de ellas?

- **Maven build** → Compila el código del proyecto
- **Maven clean** → Elimina todos los ficheros hechos por los builds anteriores
- **Maven generate-sources** → Genera código para incluirlo en la compilación
- **Maven install** → Instala los paquetes de la biblioteca en un repositorio local, compila el proyecto y lo comprueba.

Así que para nuestra prueba, vamos a elegir la opción de **Maven install**.

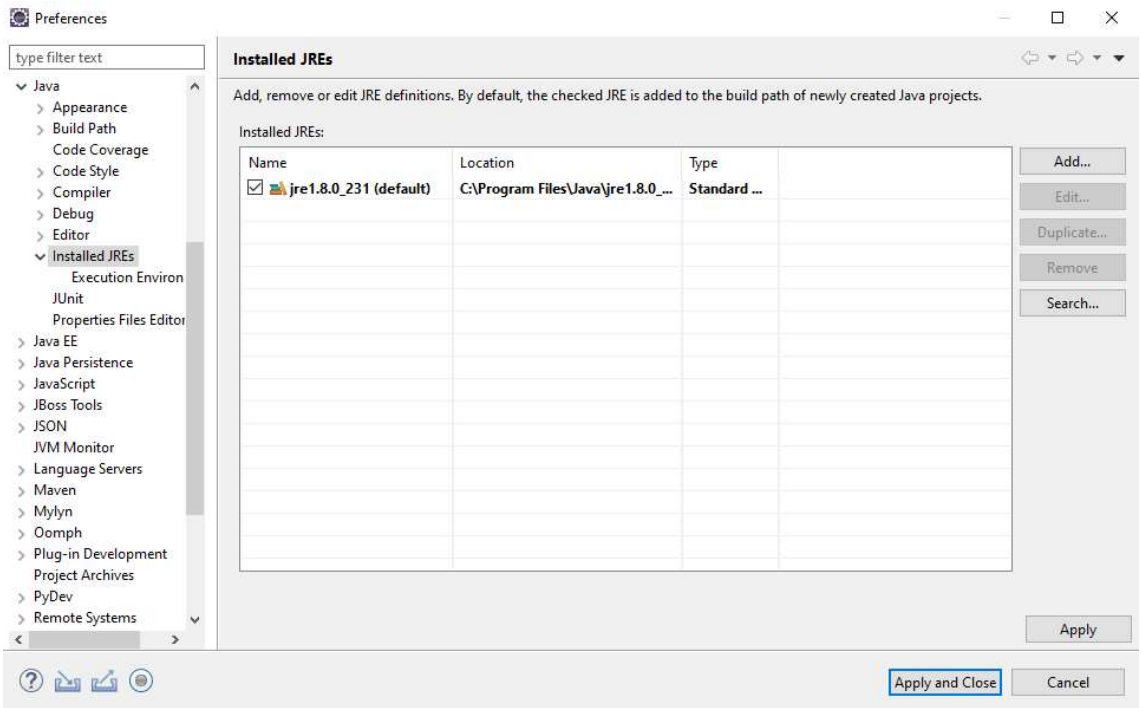


Al ejecutar Maven install se te abrirá una consola y ejecutará la creación del proyecto. Puede darte algo del estilo:

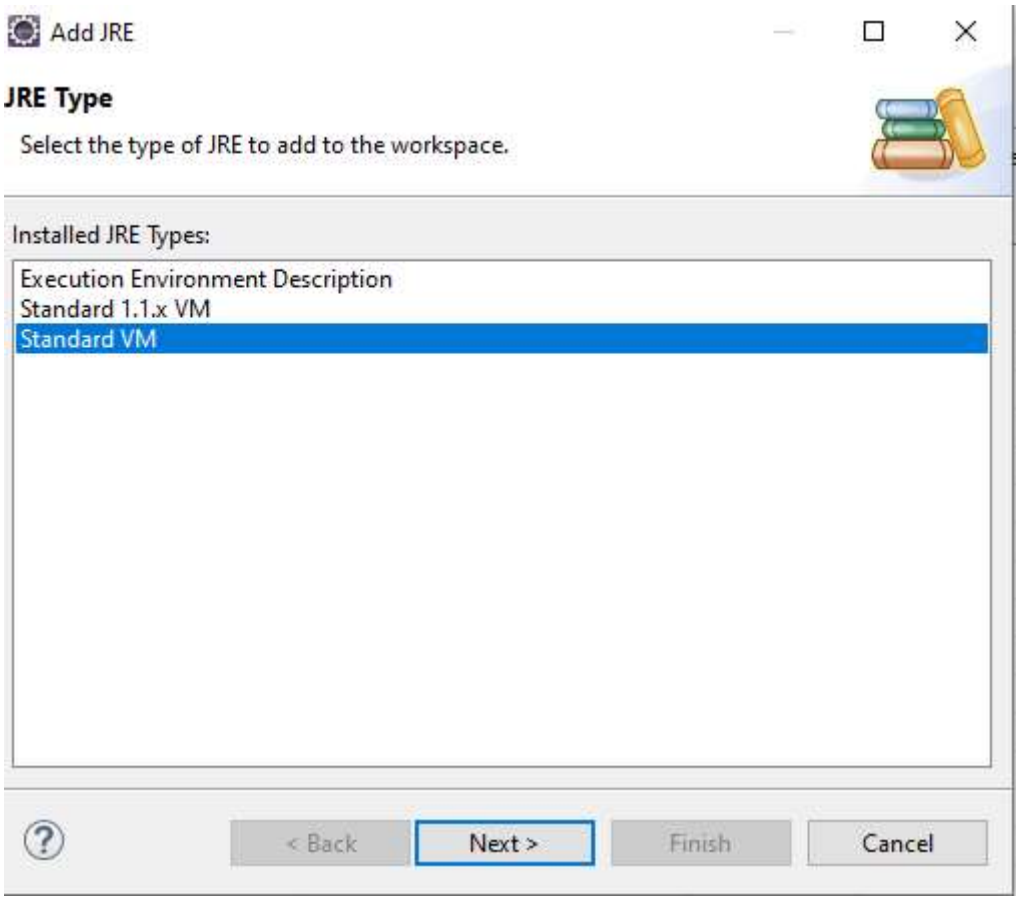
```
[INFO] Scanning for projects...
[INFO]
[INFO] --------------------< com.fran:ejemploGestorMaven >---------------------
[INFO] Building ejemploGestorMaven 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO] Downloading from : https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-install-plugin/2.4/
[INFO] Downloaded from : https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-install-plugin/2.4/ma
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ ejemploGestorMaven ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\Fran\eclipse-workspace\ejemploGestorMaven\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ ejemploGestorMaven ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to C:\Users\Fran\eclipse-workspace\ejemploGestorMaven\target\classes
[INFO] -------------------------------------------------------------
[ERROR] COMPILATION ERROR :
[INFO] -------------------------------------------------------------
[ERROR] No compiler is provided in this environment. Perhaps you are running on a JRE rather than a JDK?
[INFO] 1 error
[INFO] -------------------------------------------------------------
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  9.067 s
[INFO] Finished at: 2020-04-04T20:04:00+02:00
[INFO] ------------------------------------------------------------------------
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-compiler-plugin:3.1:compile (default-compile) on pr
[ERROR] No compiler is provided in this environment. Perhaps you are running on a JRE rather than a JDK?
[ERROR] -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException
```

Esto quiere decir que tenemos mal configurada la ruta al **JDK** en el **workspace de Eclipse**.

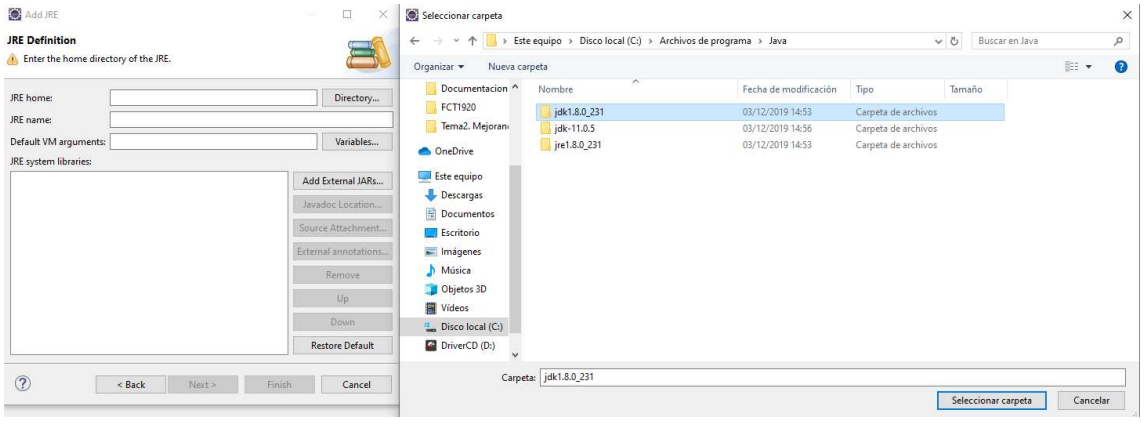
Para solucionarlo hay que ir a **Window** → **Preferences** → **Java** → **Installed JRE**. Lo que debe aparecer es la versión de **JDK**, sin embargo vemos que tenemos una **JRE**:

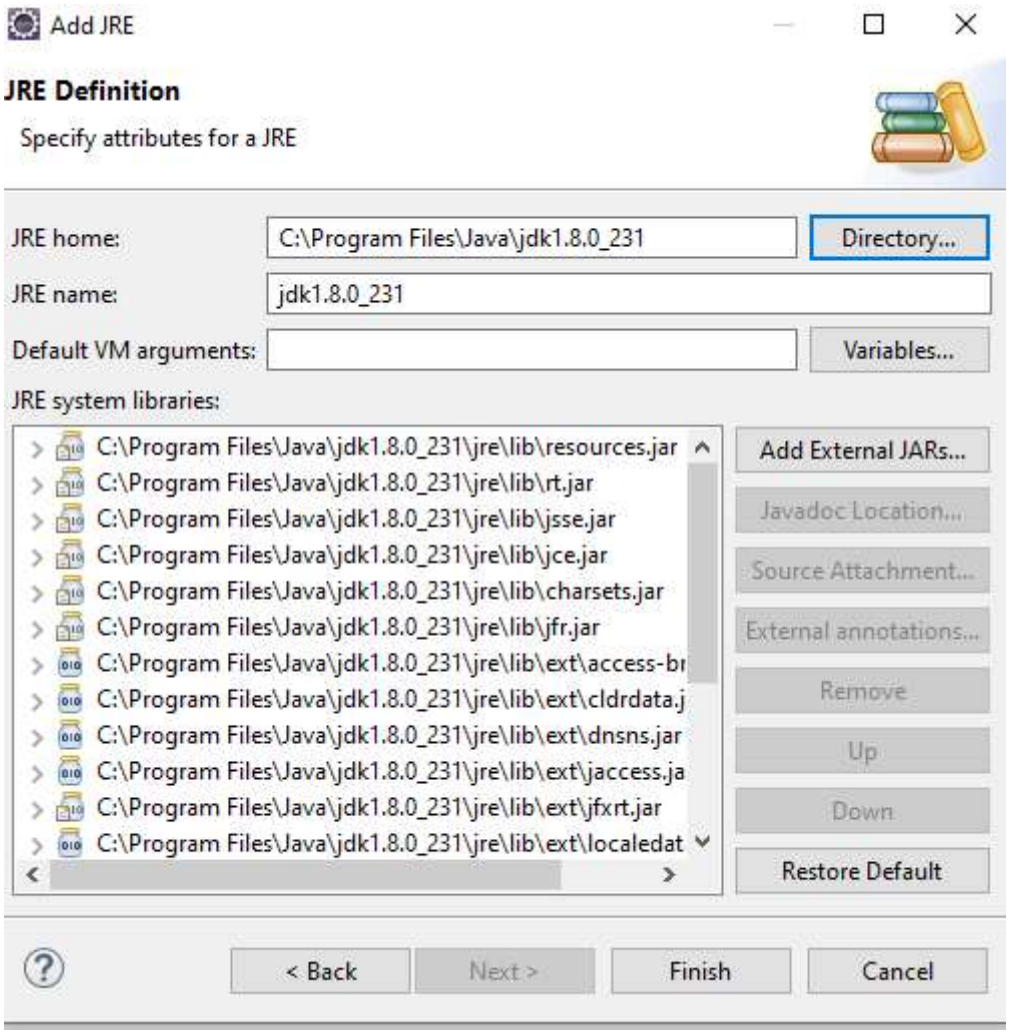


Para subsanarlo pulsamos sobre **Add** → **Standard VM** → **Directory** y elegimos la carpeta de instalación de nuestro **JDK**



Seleccionamos en directorio y buscamos el JDK que tengamos descargado, importante no seleccionar una carpeta de JRE. JDK (Java Develovpment Kit) es similar a JRE (Java Runtime Edition) pero con las herramientas necesarias para compilar y otras muchas tareas.

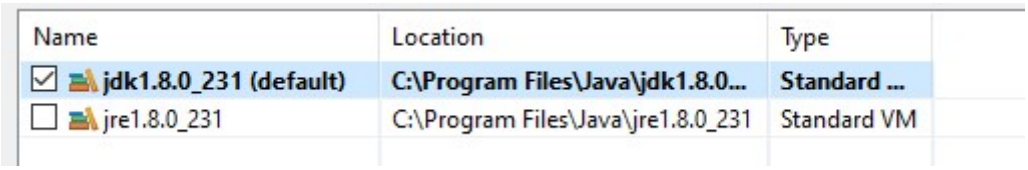




Nota: os recuerdo si no tenéis descargado un JDK que lo hagáis desde la página de Oracle, os pongo por ejemplo la última versión de Java 8:

https://www.oracle.com/java/technologies/javase-jdk8-downloads.html

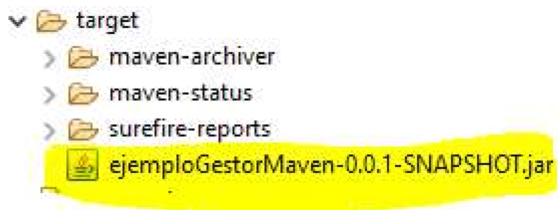
Seleccionamos el JDK en lugar del JRE y aplicamos los cambios:



Ya podemos repetir el proceso del **Maven install** habiendo corregido ese error. Ahora el resultado obtenido será este:

```
Markers  Properties  Servers  Data Source Explorer  Snippets  Console
<terminated> C:\Program Files\Java\jdk1.8.0_231\bin\javaw.exe (4 abr. 2020 20:25:50)
Running com.fran.ejemploGestorMaven.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.015 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ ejemploGestorMaven ---
[INFO] Building jar: C:\Users\Fran\eclipse-workspace\ejemploGestorMaven\target\ejemploG
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ ejemploGestorMaven ---
[INFO] Downloading from : https://repo.maven.apache.org/maven2/org/codehaus/plexus/plex
[INFO] Downloaded from : https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexu
[INFO] Downloading from : https://repo.maven.apache.org/maven2/org/codehaus/plexus/plex
[INFO] Downloaded from : https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexu
[INFO] Downloading from : https://repo.maven.apache.org/maven2/org/codehaus/plexus/plex
[INFO] Downloaded from : https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexu
[INFO] Downloading from : https://repo.maven.apache.org/maven2/org/codehaus/plexus/plex
[INFO] Downloaded from : https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexu
[INFO] Downloading from : https://repo.maven.apache.org/maven2/org/codehaus/plexus/plex
[INFO] Downloaded from : https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexu
[INFO] Downloading from : https://repo.maven.apache.org/maven2/org/codehaus/plexus/plex
[INFO] Downloaded from : https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexu
[INFO] Downloading from : https://repo.maven.apache.org/maven2/org/codehaus/plexus/plex
[INFO] Downloaded from : https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexu
[INFO] Downloading from : https://repo.maven.apache.org/maven2/org/codehaus/plexus/plex
[INFO] Downloaded from : https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexu
[INFO] Installing C:\Users\Fran\eclipse-workspace\ejemploGestorMaven\target\ejemploGest
[INFO] Installing C:\Users\Fran\eclipse-workspace\ejemploGestorMaven\pom.xml to C:\User
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  12.569 s
[INFO] Finished at: 2020-04-04T20:26:06+02:00
[INFO] ------------------------------------------------------------------------
```

Como vemos el proyecto ha sido creado correctamente. Si pulsamos con el botón derecho sobre el nombre del proyecto y refrescamos, veremos que la carpeta **target** ya no está vacía:



Ahora comprobamos que ya tenemos el empaquetado de nuestro proyecto, que será el ejecutable que podremos subir a un servidor de aplicaciones y lanzarlo como un programa independiente de Eclipse. Si quieres puedes probarlo en tu propio ordenador, accediendo por la consola a la ruta:

```
C:\Users\Fran\eclipse-workspace\ejemploGestorMaven>java -cp .\target\ejemploGestorMaven-0.0.1-SNAPSHOT.jar com.fran.ejem
ploGestorMaven.App
Hello World!
```

También puedes ejecutar el archivo como un jar, pero para ello comprueba abriendo el jar como un archivo comprimido y que en la carpeta META-INF hay un archivo llamado

MANIFEST.MF que debe contener la clase que debe ejecutarse como la principal, ya que si no nos daría un error del tipo:

```
C:\Users\Fran\eclipse-workspace\ejemploGestorMaven\target>java -jar "ejemploGestorMaven-0.0.1-SNAPSHOT.jar"
no hay ningún atributo de manifiesto principal en ejemploGestorMaven-0.0.1-SNAPSHOT.jar
```

Pero si dentro de ese archivo ponemos la clase a ejecutar cuando ejecutemos el jar:

```
Manifest-Version: 1.0
Archiver-Version: Plexus Archiver
Built-By: Fran
Created-By: Apache Maven 3.6.1
Build-Jdk: 1.8.0_231
Main-Class: com.fran.ejemploGestorMaven.App
```

Ya podremos ejecutar el comando anterior:

```
C:\Users\Fran\eclipse-workspace\ejemploGestorMaven\target>java -jar "ejemploGestorMaven-0.0.1-SNAPSHOT.jar"
Hello World!
```

Con esto, ya sabes cómo generar un jar para instalarlo en un servidor de aplicaciones como podría ser Heroku.

Para profundizar…

- Maven en la Wikipedia: https://es.wikipedia.org/wiki/Maven
- Maven en Eclipse: http://www.programandoapasitos.com/2017/07/tutorial-maven-en-eclipse.html
- Solución de errores al construir el jar: https://stackoverrun.com/es/q/2542748
- Maven Repository: https://mvnrepository.com/
- Información sobre el pom.xml: https://www.javaworld.com/article/2071772/the-maven-2-pom-demystified.html