**INSTALACION DE MAVEN**

* Una vez extraído el archivo .zip con la información solo hace falta cambiarle el nombre a Maven y pegarlo sobre C: deberá quedar C:\Maven
* Similar a Java con las variables de entorno, pero con la diferencia que el directorio contenedor va sobre raíz de mi almacenamiento, es decir. En java va en C:\Program File\Java…

Ahora es C:\Maven

* Verificando si quedo instalado con mvn -version

***¿Qué es Maven?***

Es una herramienta de gestión y comprensión de proyectos, y como tal, ayuda a gestionar nuestro proyecto.

***¿Cómo me beneficia Maven?***

Proporciona beneficios para el ciclo de desarrollo de mi proyecto, empleando convenciones y practicas estándar. Simplifica el proceso de construcción del proyecto, siendo que también lo mas destacado es que se descargan los archivos .jar de los que dependemos para hacer que nuestro proyecto funcione desde el repositorio central de Maven.

***Comenzando Proyecto Maven***

$ mvn -B archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4

//Simplificado sin especificaciones

$ mvn archetype:generate

Una vez ejecutado se creará el archivo pom.xml que contendrá toda la información base de nuestro proyecto. Contiene le modelo de objetos del proyecto. El POM es la unidad básica de trabajo en Maven. En resumen, POM contiene toda la inforamcion importante del proyecto.

En nuestro pom nos encontramos etiquetas tales como:

**project**: etiqueta de nivel superior de un archive maven

**modelVersion**: indica la versión de modelo de objeto que usa POM.

**groupId**: indica el ID del grupo que creo el proyecto, MOBY en los ejemplos

**artifactId**: indica el nombre base del artefacto principal, el artefacto principal suele ser un archivo .JAR. O es el nombre común del proyecto.

**version**: version del artefacto generado por el proyecto.

**Name**: nombre de visualizacion usado para el proyecto

**url:** no es necesario

**properties:** area de definición constante del archivo POM

**dependencies:** lista de elementos de dependencias <dependency>

**build:** Este elemento maneja cosas como declarar la estructura de directorios de su proyecto y administrar complementos.

Para realizar una compilación es necesario ejecutar el siguiente comando

$ mvn compile

Para realizar un testeo unitario:

$ mvn test

Para compilar mis fuentes de testeos/pruebas.

$ mvn test-compile

Para crear mi archivo .JAR

$ mvn package

Para instalar mi artefacto generado en mi repo local

$ mvn install

Para remover mi carpeta target con todos los datos de compilación

$ mvn clean

**Agregar Recursos a mí .JAR**

Simplemente es crear un directorio sobre nuestro directorio base llamado main, conjunto a otro llamado java - Ejemplo para su comprensión:

mi-aplicación

| - pom.xml

`- src

| - main

**| | - java**

| | `- com

| | `- Moby

| | `- app

| | `- App.java

**|** **| - resources //carpeta con sus recursos que se agregaran**

| | `- launcher.properties

| | `- META-INF

| | `- application.properties

`- test

`- java

`- com

`- Moby

`- app

`- AppTest.java

Entonces todos esos archivos dentro del directorio resources, se empaquetará en el archivo .JAR.

Si quiero agregar los recursos sobre mis pruebas unitarias es la misma situación que la anterior

En la unit test, con un simple fragmento de código puedo acceder al recurso para la prueba requerida.

// Recuperar recurso

InputStream es = getClass().getResourceAsStream("/test.properties");

// Haz algo con el recurso

**VIDEO**

¿Why do we use Maven?

La herramienta, MAVEN, se utiliza para administrar cualquier proyecto basado en JAVA. Nos ayuda a simplificar el trabajo ayudándonos sobre nuestros proyectos.

Ayuda a obtener los archivos .JAR necesarios para nuestro proyecto, y nos da la ventaja de que no necesitamos ir a la pagina oficial de cada software(dependencia), sino es acceder a mvnrepository.com.

¿What is Maven?

Es una herramienta de compilación open-source desarrollada por el Grupo Apache, para compilar, publicar e implementar varios proyectos a la vez.

Maven esta escrito en JAVA, y también se usa para compilar proyectos escritos en otros lenguajes como C#, Ruby, Scala, etc.

La herramienta es usada para compilar y administrar cualquier proyecto basado en JAVA.

POM (Project Object Model): Maven está basado en esto, y se enfoca en la simplificación y estandarización del proceso de construcción. En el proceso de construcción se ocupa de:

: [ITEMS]

**Maven Repository**

Es una especie de directorio donde tenemos todos nuestros archivos .JAR empaquetados. Son todos los archivos en una misma ubicación.

Siempre hace referencia a los directorios de archivos JAR empaquetados que contienen metadatos. Los metadatos refieren a los archivos POM.xml. Estos metadatos son los que permiten descargar las dependencias.

* Hay tres tipos de repositorios:

1. **Local**: repositorio de mi PC
2. **Remote**: el repo remoto es similar al central, pero es creado por nosotros mismos. Si no encuentra archivo, descarga e instala sobre el cache de nuestro repo local. Acá puedo almacenar todas mis dependencias particulares. Y luego puedo volver a descargarlas.

Si no hay dependencia en el repo loca, se busca en el remoto, y si esta, se descarga y se hace referencia a nuestro cache del repo local.

1. **Central**: si al buildear Maven no encuentra mi dependencia en mi repo local, las descarga desde y las pone nuevamente en el cache de mi repo local. Con esta puedo ver todas las dependencias de la comunidad.

**Conceptos Básicos de Maven**

***POM***: Project Object Model, es un archive .xml que tiene toda la información necesaria de nuestro proyecto con su configuración y de talle, de como se compilara, en que versionado, que dependencias utilizara, que versión de java tiene, etc. En sí, es una descripción del proyecto.

Este archivo esta en el directorio de inicio del proyecto. Por lo que Maven busca el POM cuando se realice cualquier tarea. Maven procesara todo lo que esta configurado sobre este archivo.

***Dependencias***: Las dependencias en sí, son nuestros archivos JAR, por eso se llaman así. Nuestro proyecto dependerá de estos archivos. Una vez que se ejecuta el POM.xml, se busca sobre mi repositorio Local, si no se encuentra tal dependencia, Maven puede ir al repositorio Remoto, o Central, descargarlo y almacenarlo en Caché.

***Build Life Cycles(***ciclo de vida de la construcción o compilación***):*** es una secuencia de fases de construcción que estas fases, tiene una secuencia de objetivos.

Cada objetivo tiene una tarea en particular. En el momento que se ejecuta una fase se compilan todos los objetivos relacionados, y sus complementos.

Son muchas fases:

Maven **Validate**: valida si el proyecto dispone de toda la información necesaria para ser procesado.

Maven **Compile**: Compila los archivos .java y generar los .class en su directorio definido.

Maven **Test**: es algo principal ya que se encarga de realizar los testeos unitarios. Asi aseguramos de que el código esta limpio.

*// UNA VEZ FINALIZADA ESTA ETAPA, SE PUEDE EMPAQUETAR*

Maven **Package**: Empaquetamos nuestro código (.class) para permitir la ejecución.

{.jar | .war | .ear}

Maven **Verify** Se lanzan los test de integración para confirmar que todo anda.

Maven **Install**: en esta fase lo que hace es desplegar mi artifact con su versionado (SNAPSHOT) en mi repositorio local.

Una vez hecho esto, puedo utilizar este SNAPSHOT.jar como si fuese una dependencia sobre otro proyecto, haciendo que el proyecto que necesite de este nuevo SNAPSHOT, consuma de él. Siempre tengo que tener en cuenta de hacer un $ mvn clean y luego el $ mvn install. Y simplemente desde el POM de nuestro nuevo proyecto, utilizar la dependencia. Llamando al paquete del mismo.

<groupId>com.Moby</groupId>

<artifactId>TestMaven</ artifactId >

<version>SNAPSHOT-1.0</version>

Maven **Deploy:** Cuando se hace un deploy, lo que se genera es nuestro artifact en nuestro repositorio local, y luego se sube en un repositorio remoto, este ultimo se hace con la finalidad de que si otro Developer quiere acceder a mi ultima versión, pueda utilizar esa dependencia sin problema.

***Build Profile:*** Es un perfil de compilación, por lo general si no especificamos nada acerca de cómo será la compilación, Maven al leer el POM utilizará el perfil por default. Con esto podemos establecer reglas en nuestra compilación para diferentes entornos.

***Build Plugins:*** ¿?

**Arquitectura de Maven**

Maven lo que hace es leer nuestro archivo POM, entonces al leer, procesara los componentes, descargando principalmente las dependencias ya sea del central o remote repository. Luego crea y genera reporte acorde a los requerimientos, y ejecutando life-cycle, phases, goal, plugin, etc.

Y al final, termina generando nuestro artifact o SNAPSHOT en nuestro repositorio local.

Ejecutar proyecto desde Maven luego de usar mvn compile:

mvn exec:java -Dexec.mainClass="com.mycompany.app.NAME\_MAINFILE"

Si usamos directamente este puglin en el POM

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec.maven-plugin</artifactId>

<version>3.0.0</version>

<configuration>

<mainClass>com.mycompany.app.NAME\_MAINFAIL</mainClass>

</configuration>

</plugin>

En consola solo podemos ejecutar mvn exec:java