P0-Introducción a Maven

Objetivos

- Iniciación en el uso de Maven: arquetipos, estructura de un proyecto, fichero pom.xml, compilación, empaquetado y ejecución con Maven

Herramientas

- Maven 3.0.4

Maven es una herramienta de construcción de proyectos Java (similar a Ant o Make). Por lo tanto automatiza los procesos de compilar, probar, empaquetar, desplegar.... un proyecto Java.

En este curso utilizaremos Maven (versión 3.0.4), tanto desde línea de comandos (comando mvn) como desde un IDE (NetBeans 7.2). En esta sesión, vamos a comenzar a usar Maven desde línea de comandos.

Ejercicios

1. **Ejercicio 1**. Vamos a ver cómo **CREAR** un proyecto java a partir de un arquetipo. Un arquetipo es un proyecto Maven que sirve como "plantilla" para iniciar el desarrollo de un proyecto Java siguiendo el estándar que propone Maven. Comenzaremos por trabajar con el **arquetipo maven-archetype-quickstart**, que genera un proyecto java básico. Para ello tecleamos, desde línea de comandos (por ejemplo desde el directorio $HOME):

```
mvn archetype:generate -DgroupId=prueba -DartifactId=prueba \
-DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

> **Nota**: El carácter "\" al final de la primera línea NO forma parte del comando. Dicho carácter indica que, cuando tecleéis el comando desde un terminal, tenéis que escribir TODO en una única línea.
>
> En este caso concreto el comando **mvn** ejecuta la "goal" archetype:generate (explicaremos el concepto de *goal* más adelante) y añade valores concretos para determinados parámetros utilizando la sintaxis -Dnombre_variable=valor

Al ejecutar el comando anterior, como resultado, se habrá creado el directorio "prueba", que contiene un nuevo proyecto Maven java. Nuestro proyecto (contenido del directorio prueba) está formado por un conjunto de artefactos (ficheros de cualquier tipo generados durante el desarrollo del proyecto), entre los cuales hay uno que es imprescindible y que caracteriza a cualquier proyecto Maven: es el fichero pom.xml, del cual hablaremos más adelante. Todos los proyectos Maven almacenan los artefactos siguiendo una estructura de directorios estándar (propia de Maven). Se pide:

A) Crea un esquema con la estructura de directorios y todos los ficheros que se han generado

B) Edita el fichero App.java e indica cuál es su contenido

C) Crea otro proyecto Maven utilizando el comando anterior (desde $HOME), pero cambiando los parámetros, en este caso utiliza groupId=nombreGrupo, y artifactId=proyectoMaven2. Observa e indica las diferencias entre los dos proyectos creados.

D) Crea un tercer proyecto Maven con el comando: `mvn archetype:generate`. Este comando es una forma alternativa de utilizar el plugin archetype (veremos más

adelante el concepto de *plugin* en Maven). En este caso, en lugar de indicar nosotros en la línea de comando los valores de los parámetros, el sistema nos preguntará,de forma **INTERACTIVA** dichos valores, que iremos introduciendo por teclado. Lo primero que tenemos que decidir es el arquetipo a utilizar (es decir el valor de la variable archetypeArtifactId). Los arquetipos están numerados, y el arquetipo por defecto es justamente "maven-archetype-quickstart", por lo que aceptamos el número que nos aparece por defecto. Podemos elegir la versión del arquetipo, normalmente siempre elegiremos la última versión (que además, nos propone por defecto). Como "groupId" indicamos el valor "ppss". Como "artifactId" elegimos "proyectoMaven3". Dejamos el identificador de versión por defecto ("1.0-SNAPSHOT). Como nombre de paquete ponemos "paquete", y confirmamos la configuración. Observa e indica las diferencias con los proyectos que has creado anteriormente.

2. **Ejercicio 2**. Ahora vamos a trabajar con los proyectos creados. Dichos proyectos, aunque son simples plantillas que modificaremos para incluir nuestro código particular, son proyectos COMPLETOS que pueden compilarse, probarse, empaquetarse, ejecutarse... Cada una de las acciones que acabamos de mencionar, constituyen lo que se denominan "fases del ciclo de vida de Maven", y siguen un orden predeterminado (por ejemplo, la compilación se realiza previamente a las pruebas, el empaquetado es posterior a las pruebas...)
   A) Vamos a **COMPILAR** el primer proyecto. Para ello (desde el directorio "prueba") ejecuta el comando `mvn compile`. Revisa el esquema que has hecho inicialmente con la estructura de directorios y ficheros generados e indica y anota las diferencias. ¿Qué ha ocurrido al ejecutar el comando `compile`? Haz lo mismo con los otros dos proyectos y analiza las diferencias con los artefactos generados en el ejercicio anterior.
   B) Por pantalla te aparece la información de los pasos (fases) que va realizando Maven. Indica y anota dichos pasos.
   C) Ahora vamos a **EMPAQUETAR** los proyectos. Ya te habrás dado cuenta de que para ejecutar cualquier acción Maven sobre nuestro proyecto, tenemos que estar en el directorio raíz del mismo. Para empaquetar utiliza la orden `mvn package`. Fíjate que en todos los casos se genera un fichero (artefacto) **.jar**
   D) Ahora vamos a **INSTALAR** los proyectos. El proceso de instalación consiste en copiar el "jar" generado en un repositorio local de Maven. Este repositorio, por defecto se crea en $HOME/.m2 y contiene cualquier fichero ".jar", ".war", ... que Maven genera y/o puede necesitar utilizar para construir los proyectos. Puedes consultar el contenido del repositorio local para ver qué ficheros contiene, verás, por ejemplo, que contiene el fichero `repository/junit/junit/3.8.1/junit-3.8.1.jar` (este fichero es el que aparece referenciado en nuestro fichero pom.xml, dentro de la etiqueta `<dependencies>`). Cuando ejecutamos Maven, si se necesita alguna librería para construir el proyecto, primero Maven consulta en el repositorio local. Si en el repositorio local no se encuentra el fichero, entonces Maven accede a un repositorio remoto, para descargarse dicho fichero. De esta forma , nuestro repositorio local irá creciendo a medida que construyamos más proyectos. Si borramos el directorio .m2, la próxima vez que ejecutemos Maven, se volverán a descargar los ficheros necesarios. Para instalar el primer proyecto ejecuta (desde $HOME/prueba) el comando mvn install. ¿Dónde se ha copiado exactamente el fichero "jar" del proyecto prueba generado por Maven? Instala los otros dos proyectos.
   E) Ahora vamos a **EJECUTAR** nuestro primer proyecto con el comando `mvn exec:java -Dexec.mainClass="prueba.App"`. Verás el resultado de la ejecución en el terminal. Ejecuta los otros dos proyectos.

3. **Ejercicio 3**. Finalmente vamos a revisar el contenido del fichero **pom.xml**, que podemos ver en el directorio raiz de CUALQUIER proyecto Maven. De hecho, cuando ejecutamos el comando **mvn**, éste sigue las indicaciones del fichero pom.xml, que debe encontrarse necesariamente en el directorio desde el que ejecutamos dicho comando. El fichero pom.xml almacena la configuración de nuestro proyecto Maven, y básicamente contiene cuatro tipos de información: (a) información general del proyecto (nombre del proyecto, url del proyecto, lista de desarrolladores...), (b) la configuración de la construcción (podemos cambiar el comportamiento por defecto y/o añadir nuevas acciones (*goal plugins*)), (c) configuración del entorno (para ello utilizaremos *profiles*, que pueden activarse para usar en diferentes entornos de trabajo), y (d) relaciones con otros proyectos (podemos utilizar relaciones de herencia entre proyectos, o de agregación) .

   A) Edita el fichero pom.xml del proyecto prueba, y observa y explica la información que almacenan las etiquetas: *groupId*, *artifactId*, *packaging*, *version*, *name*, *url (*ver http://maven.apache.org/pom.html#Maven_Coordinates y http://maven.apache.org/pom.html#More_Project_Information*)* Nuestro proyecto tiene declarada una dependencia con la librería junit 3.8.1. Es una librería necesaria para automatizar las pruebas, y más adelante veremos cómo usar junit, aunque nosotros utilizaremos la versión 4.8.2.

   B) A continuación vamos a "agregar" nuestros tres proyectos en un único proyecto maven de tipo **MULTI-MÓDULO**, es decir, un proyecto que contiene múltiples proyectos Maven. Para ello utilizaremos el **arquetipo pom-root**, de la siguiente forma (nos situaremos en el directorio $HOME):

```
mvn archetype:generate -DgroupId=ppss -DartifactId=miMultiModulo \
-DarchetypeGroupId=org.codehaus.mojo.archetypes \
-DarchetypeArtifactId=pom-root -DinteractiveMode=false
```

   C) Si editamos el fichero pom.xml veremos que ahora el empaquetado no es "jar" sino "pom". Vamos a añadir un primer módulo a nuestro proyecto miMultiModulo. Concretamente nuestro primer proyecto prueba. Para ello, simplemente añadimos las etiquetas `<modules><module>../prueba</module></modules>`. Para comprobar que hemos añadido el módulo `prueba` ejecutaremos `mvn clean install` (desde el directorio que contiene nuestro proyecto multi-módulo) ¿Puedes explicar cuáles son los efectos de ejecutar el comando anterior?

> **Nota**: El parámetro `clean` lo utilizaremos para ejecutar el ciclo de vida de Maven "clean". ¿Cuáles son los efectos de esta acción? Dedúcelo tú mismo ejecutando `mvn clean` desde cualquier proyecto maven que previamente hayas compilado, y observa los efectos sobre los artefactos generados en el directorio de dicho proyecto.

   D) A continuación añade en nuestro proyecto multi-módulo los otros dos proyectos que hemos creado: *proyectoMaven2* y *proyectoMaven3*

   E) Como habrás comprobado, las acciones sobre el proyecto multi-módulo se realizan sobre todos los módulos que éste contiene. Ahora mismo, debes tener en tu directorio $HOME cuatro proyectos: *prueba*, *proyectoMaven2*, *proyectoMaven3* y *miMultiModulo*. Si quisiéramos copiar los proyectos para, por ejemplo, seguir trabajando en casa, fíjate que tendríamos que copiar los cuatro directorios tal cual en el ordenador de casa. Imagínate que has creado otros 5 proyectos maven más en el mismo directorio. Tendrías que acordarte exactamente de qué proyectos forman parte del proyecto *miMultiModulo*. Una forma mejor de trabajar es situar todos los proyectos que forman

parte del proyecto multi-módulo en el directorio raiz de dicho proyecto. En este caso tendrías que actualizar el pom.xml para modificar la ruta actual de los proyectos agregados. Hazlo así, y de esta forma sólo tendremos que copiar el directorio *miMultiModulo* (y todos sus subdirectorios, claro). Consulta este enlace para ver una forma alternativa de crear, desde el principio, un proyecto multi-módulo (http://anadreamy.wordpress.com/2012/03/27/como-crear-un-proyecto-multimodulo-con-maven/)