Taller de Programación Web

MAVEN

¿QUÉ ES MAVEN?

Maven es una herramienta open source para administrar proyectos de software. Por administrar, nos referimos a gestionar el ciclo de vida desde la creación de un proyecto en un lenguaje dado, hasta la generación de un binario que pueda distribuirse con el proyecto.

Maven nos brinda una estructura consistente de proyectos (todos los proyectos Maven tienen por default los mismos directorios) y herramientas necesarias actualmente para los proyectos de software: gestión avanzada de dependencias, informes sobre testing automáticos y extensibilidad vía plugins.

REQUISITOS

- Descargar Maven: http://maven.apache.org/download.html
- Debe estar definida la variable de entorno **JAVA_HOME** en donde se encuentra la instalación del JDK que queremos usar y esta variable agregarla a la variable **PATH** (Consultar los videos de instalación de JAVA en Windows y Linux, para hacerlo de manera similar).
- Para comprobar que esté correctamente instalado ejecutar en una consola: `mvn -v` (nos deberá devolver la versión sin mensajes de error).

CREACIÓN DE PROYECTOS MAVEN

POM son las siglas de "Project Object Model" (Modelo de Objetos de Proyecto), ya hemos dicho que Maven es una herramienta de administración de proyectos. Un POM no es más que la abstracción usada por Maven para definir dichos proyectos, como tal contiene los atributos de estos y las instrucciones para construirlo.

La ejecución de un archivo POM siempre genera un "**artefacto**". Este artefacto puede ser cualquier cosa: un archivo jar, un archivo war, un archivo zip o el mismo archivo pom.

Maven trabaja por medio de la modularización de proyectos. De esta forma tendremos varios módulos que conforman un sólo proyecto. Digamos, que queremos realizar más de una aplicación web para nuestra organización, que poseerán responsabilidades distintas (facturación, otra adm. de stock de productos, etc) pero pueden compartir clases y servicios en común. Para no copiar dichas clases cada vez que las necesitemos podemos crear un proyecto maven **padre** o **componente**, de esa forma podremos heredarlo o incluirlo en el proyecto deseado.

CONVENCIONES

- El nombre de la carpeta debe ser igual al nombre del proyecto (en el caso de proyectos módulo, esto es obligatorio, si no son módulos, sino proyectos padre esto no es forzoso pero se recomienda)
- A nivel raíz de la carpeta debe estar el archivo pom.xml

ATRIBUTOS

- **groupId**: Representa el paquete de proyecto. Típicamente aquí se pone el nombre de tu empresa u organización, ya que conceptualmente todos los proyectos con ese groupId pertenecen a una sola empresa. La definición del grupo por convención es en sentido inverso al del dominio de la empresa u organización del proyecto. Ej: Si trabajamos para la organización **ACME (acme.com)** definiremos nuestro grupo como **com.acme**
- **artifactId**: Es el nombre de tu proyecto. Ej: ecommerce, blog, reservas, etc.
- **version**: Número de versión de tu proyecto. Con cada mejora o creación de nuevas funcionalidades se podrá cambiar manualmente o de forma automática (con script o servicios) la versión de nuestro proyecto.
- **packaging**: Paquete base donde irá tu código fuente.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-parent</artifactId>
     <version>2.5.5</version>
     <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.informatorio</groupId>
  <artifactId>e-commerce</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>e-commerce</name>
  <description>Demo project for Spring Boot</description>
  <properties>
     <java.version>11</java.version>
  </properties>
  <dependencies>
     <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
     </dependency>
  </dependencies>

  <build>
     <plugins>
        <plugin>
           <groupId>org.springframework.boot</groupId>
           <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
     </plugins>
  </build>
</project>
```

Ejemplo de archivo pom.xml - Figura 1

CICLO DE VIDA

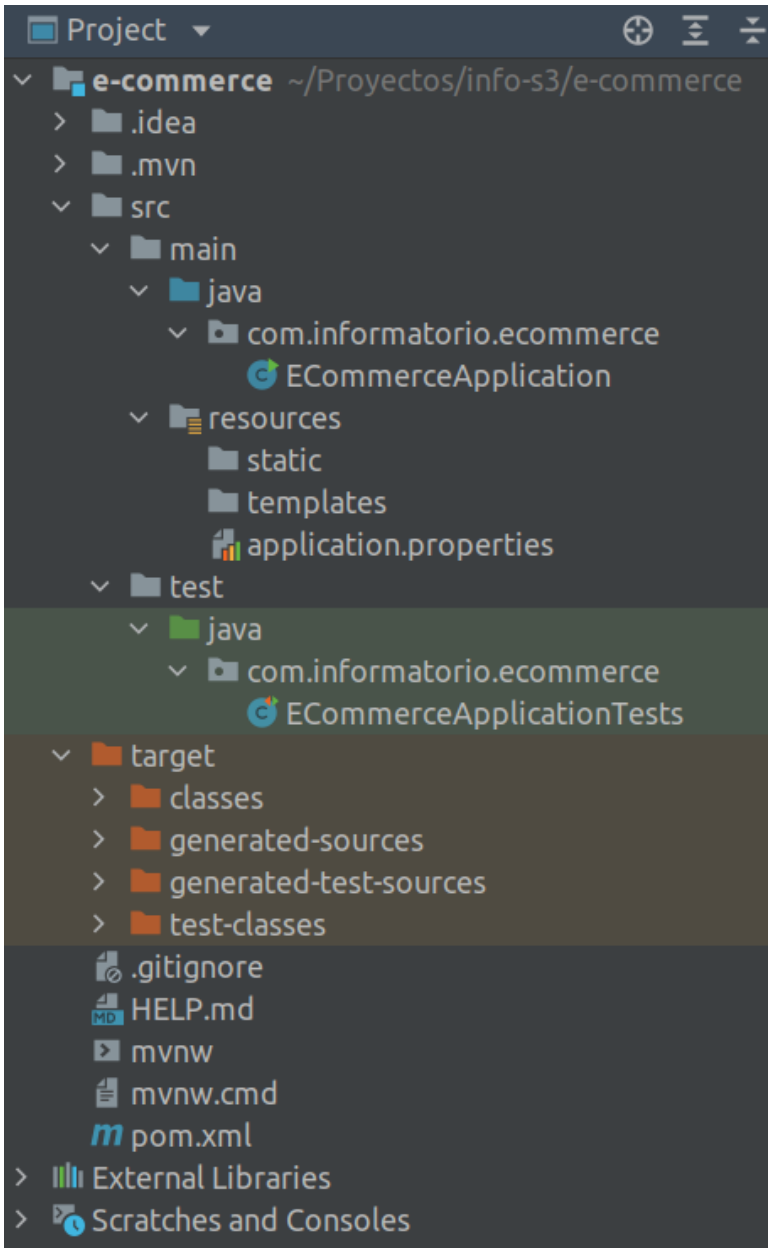
Existen 3 ciclos de vida en Maven:

- **clean**. Elimina las clases compiladas y los archivos binarios generados del proyecto.
- **default**. Genera los archivos binarios (artefactos) de nuestro proyecto
- **site**. Genera archivos html que describen nuestro proyecto

Para ejecutarlos, excepto el default, se debe de poner mvn ciclo (ej: `mvn clean` o `mvn site`). Cada ciclo de vida está constituido por varias fases. Las fases del ciclo de vida default son 24, pero mencionaremos las más relevantes:

Fase	Descripción
validate	Valida el proyecto.
initialize	Configura propiedades y crea directorios.
compile	Compila el código fuente del proyecto.
test	Ejecuta las pruebas.
package	Genera el artefacto del proyecto.
verify	Verifica el artefacto generado.
install	Instala el artefacto en el repositorio local.
deploy	Sube el artefacto a un repositorio Maven en la red.

ESTRUCTURA DE PROYECTO



Maven crea las carpetas:

- **src/main/java**: Donde se colocara el código fuente.
- **src/test/java**: Es donde residirán los test.

Maven solo incluira en el artefacto final (luego de realizar un build) solo las clases que se encuentran dentro de src/main/java

DEPENDENCIAS

En el ejemplo de archivo pom.xml (Figura 1) que vimos anteriormente podremos encontrar la etiqueta dependencies dentro se incluyen todas las librerías externas que necesitará nuestro proyecto para funcionar.

En el ejemplo podremos ver que se incluye la librería spring-boot-starter-test:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
```

Algo interesante de esta dependencia es que tiene un <scope> definido como de test. Esto indica a Maven que esta librería sólo se usará durante la fase de testing y no será incluida en los binarios (artefacto final). Las dependencias que no llevan este scope es porque queremos que sean incluidas en el binario.

Existen 6 scopes distintos para las dependencias de Maven:

Scope	Descripción
compile	Es el scope por defecto si no se especifica. Estas dependencias se usan en el classpath del proyecto y serán incluidas en el artefacto final.
provided	Estas dependencias se usan durante la fase compile y test. Pero no se incluye en el artefacto final. Porque ya se encuentran en el servidor Java, por lo que no es necesario volverlas a incluir en tu archivo war.
runtime	Indica que la dependencia será necesaria durante la ejecución de tu aplicación pero no al compilar.
test	Indica que la dependencia sólo es necesaria para compilar y ejecutar los tests del proyecto. Estas dependencias no serán incluidas en el artefacto final.
system	Igual a provided pero aquí debes especificar el path de tu disco duro al jar que contiene esta dependencia. Así evitas que Maven la busque en los repositorios.
import	Solo funciona en Maven 2.0.9 y superior. Permite importar otros archivos pom para simular herencia múltiple ya que Maven solo permite heredar de un solo archivo POM.