UTN
Facultad Regional Córdoba

TUP

Tecnicatura Universitaria en Programación

LABORATORIO DE COMPUTACIÓN III

Anexo N°3:

Maven

Material de Estudio

2° Año – 3° Cuatrimestre

Índice

.

Maven

Maven es una herramienta de gestión de proyectos de software que se utiliza principalmente para la construcción y gestión de proyectos Java. Maven proporciona una forma estandarizada de construir, empaquetar y desplegar aplicaciones, y ayuda a los desarrolladores a manejar dependencias de manera eficiente. En lugar de descargar y configurar manualmente las dependencias, Maven hace todo ese trabajo por ti.

Instalación y configuración de Maven

Para instalar Maven, usaremos ***SDKMAN*** siguiendo estos pasos:

1. Abre una terminal en tu sistema operativo basado en Unix o GitBash para Windows.
2. Verificamos que SDKMAN esté instalado con el comando "***sdk version***". Si aparece un cartel que nos dice que existe una versión más reciente, ingresamos "***Y***" para actualizar.

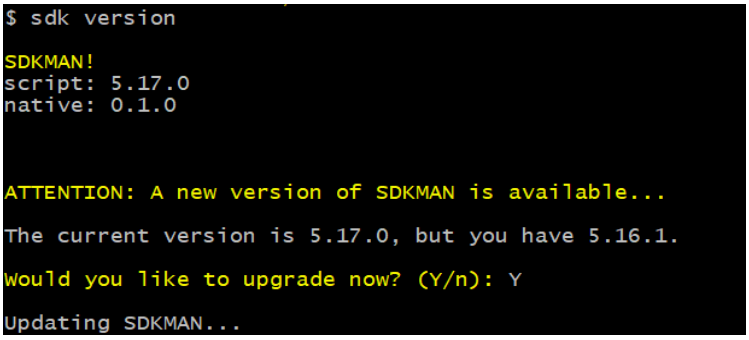

Imagen 1: Elaboración propia.

3. Con el comando "***sdk list maven***" veremos la lista de versiones de Maven que hay disponible para instalar. Veremos una lista como esta:

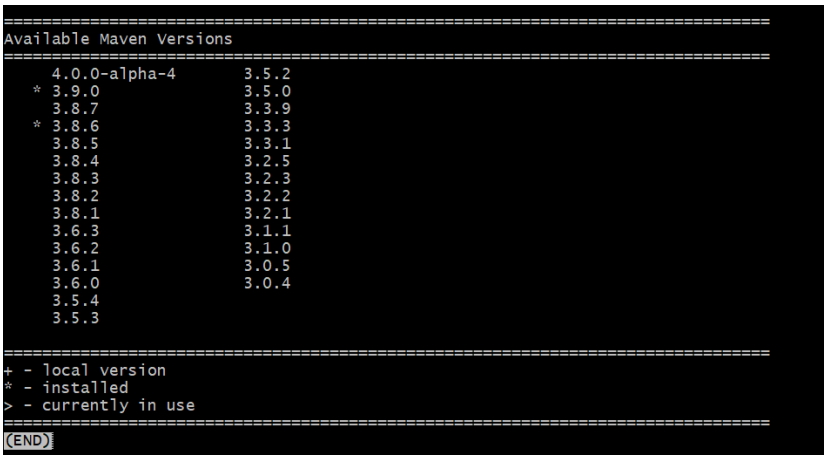

Imagen 2: Elaboración propia.

Las versiones que ya tengamos instaladas aparecerán con un asterisco al lado, como se ve en la imagen, la versión 3.9.0 y 3.8.6 ya estan instaladas.

4. En nuestro caso vamos a usar la versión estable más reciente (***3.9.0***). Para instalarla, ejecutaremos el comando "***sdk install maven 3.9.0***".

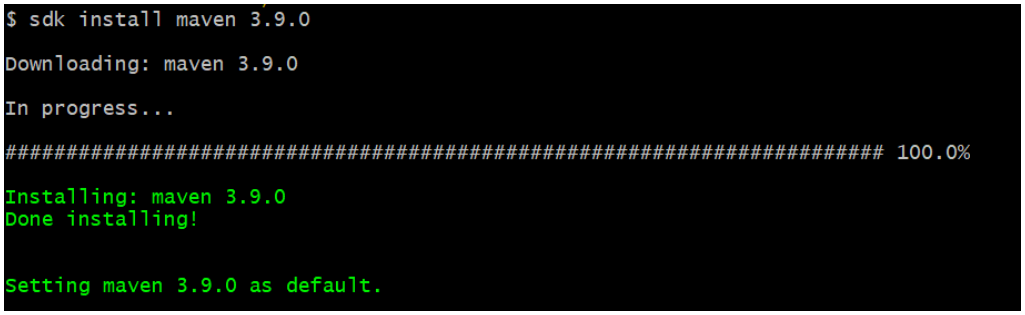

Imagen 3: Elaboración propia.

5. Para verificar la instalación, ejecutaremos el comando "***mvn -version***". Si todo salió bien, verán un mensaje como este:

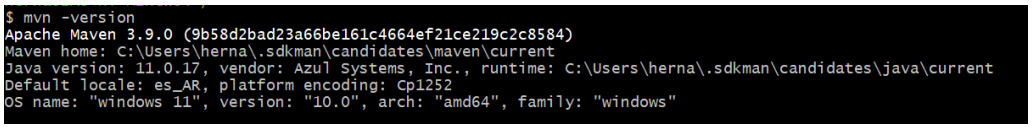

Imagen 4: Elaboración propia.

Normalmente después de instalar Maven, hay algunas configuraciones que debes hacer para poder usarlo correctamente. Con sdkman, estas configuraciones no son necesarias, ya que la herramienta las hizo por nosotros. Igualmente, a medida que vayamos usando la herramienta y dependiendo del contexto en el que la usemos, podemos requerir configuraciones adicionales, por ejemplo, configurar un repositorio remoto privado de la empresa para la que trabajamos. Cuando esto suceda, siempre puedes referirte a la documentación oficial de la herramienta para ver cómo hacerlo (https://maven.apache.org/)

Conceptos básicos de Maven

Maven es una herramienta de gestión de proyectos que se utiliza principalmente para construir, empaquetar y distribuir software. Algunos de los conceptos básicos de Maven son:

- **Proyecto**: Un proyecto es una unidad fundamental de trabajo en Maven. Representa **una colección de archivos y directorios que juntos forman una aplicación, una biblioteca o cualquier otro tipo de software** que se esté desarrollando.
- **POM (Project Object Model):** El POM es **un archivo XML que contiene información sobre el proyecto, como su nombre, descripción, versión, dependencias, plugins, etc.** Es el archivo principal de configuración de Maven y se utiliza para definir el ciclo de vida del proyecto.
- **Dependencias**: Las dependencias son archivos JAR o **bibliotecas que el proyecto necesita para funcionar correctamente**. Maven resuelve automáticamente las dependencias de un proyecto y las descarga de los repositorios remotos o las instala en el repositorio local.
- **Repositorios**: Los repositorios son **lugares donde Maven almacena las dependencias y los plugins**. Maven utiliza dos tipos de repositorios: locales y remotos. El repositorio **local es donde se almacenan las dependencias y los plugins descargados en tu sistema local**, mientras que el repositorio remoto es donde se almacenan los artefactos que no se encuentran en el repositorio local.
- **Ciclo de vida del proyecto**: El ciclo de vida del proyecto es **una secuencia de fases que se ejecutan en un orden específico para construir, empaquetar y distribuir el software**. Maven define tres ciclos de vida principales: el ciclo de vida de construcción, el ciclo de vida de despliegue y el ciclo de vida de sitio.
- **Goal**: Los goals (objetivos) son **comandos que se ejecutan dentro de una fase específica del ciclo de vida**. Los objetivos de Maven se utilizan para realizar tareas específicas, como compilar el código fuente, ejecutar pruebas, empaquetar el código en un archivo JAR o WAR, instalar el paquete en el repositorio local, desplegar el paquete en un servidor, generar informes, entre otros.
- **Plugins**: Los plugins **son componentes de software que se utilizan para realizar tareas específicas en el proyecto**. Maven utiliza plugins para ejecutar las diferentes fases del ciclo de vida del proyecto, como compilar el

código fuente, ejecutar pruebas, generar documentación, etc. Maven incluye muchos plugins integrados, pero también es posible desarrollar plugins personalizados.

Ciclo de vida

El ciclo de vida de Maven es una secuencia de fases que se ejecutan en un orden específico para construir, empaquetar y distribuir el software. El ciclo de vida está definido en el archivo pom.xml y puede ser personalizado para adaptarse a las necesidades específicas del proyecto.

A continuación, describiré cada una de las fases del ciclo de vida de construcción:

- **clean:** Se utiliza para eliminar los archivos generados en una compilación anterior
- **validate:** Esta fase se utiliza para validar que el proyecto es correcto y que todas las dependencias están disponibles. En esta fase verifica que el proyecto tenga una estructura válida y que todos los archivos de configuración necesarios estén presentes.
- **compile:** En esta fase compila el código fuente del proyecto y genera los archivos .class.
- **test:** Esta fase se utiliza para ejecutar las pruebas unitarias del proyecto. Maven ejecuta todas las pruebas en el directorio src/test y genera un informe de resultados.
- **package:** Esta fase empaqueta el código compilado y los recursos en un archivo JAR o WAR, dependiendo del tipo de proyecto.
- **verify:** En esta fase verifica que el paquete generado cumpla con los criterios de calidad definidos para el proyecto. Esto puede incluir pruebas adicionales, análisis de código, validación de licencias, etc**.**
- **install:** Esta fase instala el paquete en el repositorio local de Maven para su uso en otros proyectos locales.
- **deploy:** En esta fase copia el paquete generado al repositorio remoto o lo despliega en un servidor.
- **site:** En esta fase, Maven genera el sitio web de documentación del proyecto.

Archivos de configuración

Los archivos de configuración de Maven son una parte fundamental de su funcionamiento. Maven utiliza diferentes archivos de configuración para controlar su comportamiento y personalizar el proceso de construcción de proyectos. Estos archivos permiten definir la estructura del proyecto, las dependencias, los plugins, la configuración de compilación, entre otras cosas.

Aprenderemos sobre los diferentes archivos de configuración que utiliza Maven y su función en el proceso de construcción de proyectos. Además, veremos cómo personalizar y configurar estos archivos para adaptarlos a las necesidades específicas de cada proyecto. Conocer los archivos de configuración de Maven es esencial para aprovechar al máximo su potencial y mejorar la eficiencia en la construcción de proyectos.

Archivo pom.xml

El archivo **pom.xml** es el archivo de configuración principal de Maven. Contiene información sobre el proyecto, como el nombre, la versión, la descripción y los desarrolladores. También define las dependencias, plugins y perfiles que se utilizan en el proceso de construcción. Además, define la estructura del proyecto. Por ejemplo, puede especificar la ubicación de los directorios de código fuente y recursos, los directorios de salida para los archivos compilados y empaquetados, y los archivos que deben incluirse en el paquete generado.

Archivo settings.xml

El archivo **settings.xml** es un archivo **de configuración global que se utiliza para personalizar la configuración de Maven**. Este archivo se encuentra en el directorio .m2 de la carpeta de usuario y se utiliza para configurar opciones como los repositorios Maven que se utilizan, los servidores proxy, las credenciales de acceso a repositorios privados y la configuración de los plugins.

Archivo .m2/settings.xml

El archivo **.m2/settings.xml** es un archivo de configuración **específico de usuario** que se utiliza para personalizar la configuración de Maven para un usuario específico.

Gestión de dependencias

La gestión de dependencias es una parte fundamental en el proceso de construcción de proyectos en Maven. **Las dependencias son bibliotecas o módulos de código que se utilizan en el proyecto y que no son desarrollados por el equipo que está trabajando en él**. La gestión de dependencias consiste en administrar estas bibliotecas para asegurar que todas las dependencias necesarias para el proyecto estén disponibles y sean compatibles entre sí.

Aprenderemos sobre la gestión de dependencias en Maven. Veremos cómo se definen las dependencias en el archivo pom.xml, cómo se descargan automáticamente las dependencias de los repositorios públicos o privados, cómo se resuelven conflictos entre dependencias y cómo se excluyen dependencias no deseadas. Conocer la gestión de dependencias de Maven es fundamental para cualquier desarrollador que trabaje con esta herramienta, ya que permite garantizar la correcta construcción y ejecución de los proyectos, y evita problemas como incompatibilidades entre bibliotecas o versiones incorrectas de las mismas. Además, la gestión de dependencias automatizada de Maven permite ahorrar tiempo y esfuerzo en la administración manual de bibliotecas y módulos.

Agregar una dependencia

La gestión de dependencias se realiza a través del archivo **pom.xml**. Para agregar una dependencia al proyecto, es necesario incluir su información en el archivo. Esta información incluye **el identificador de la dependencia, que consta del grupo, el artefacto y la versión**.

- El grupo es el identificador del proyecto al que pertenece la dependencia
- El artefacto es el nombre de la biblioteca o módulo
- La versión indica la versión específica de la dependencia a utilizar.

Por ejemplo, si se desea agregar la dependencia de la biblioteca de log4j, el código a incluir en el archivo pom.xml sería el siguiente:

```
<dependencies>
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.17.1</version>
  </dependency>
</dependencies>
```

Imagen 5: Elaboración propia.

Al construir el proyecto, Maven buscará esta dependencia en los repositorios configurados y la descargará automáticamente si es necesario. Los repositorios son lugares donde se almacenan las dependencias, y pueden ser públicos o privados. Maven utiliza por defecto el repositorio central de Maven (https://search.maven.org/) para buscar dependencias públicas, aunque se pueden configurar otros repositorios adicionales.

Es posible que existan **conflictos entre dependencias, es decir, que dos o más dependencias requieran versiones diferentes de la misma biblioteca o módulo**. Para resolver estos conflictos, Maven utiliza un sistema de resolución de dependencias que selecciona automáticamente la versión más adecuada y compatible para todas las dependencias requeridas.

Excluir una dependencia

En algunos casos, es posible que se desee excluir una dependencia específica, ya sea porque no es compatible con el proyecto o porque se desea utilizar otra versión en su lugar. Para ello, se puede especificar la dependencia a excluir en el archivo pom.xml utilizando la etiqueta **exclusion** dentro de la dependencia correspondiente.

Aquí hay un ejemplo del uso de la etiqueta **exclusion** en una dependencia en el archivo pom.xml. Supongamos que en nuestro proyecto necesitamos utilizar la biblioteca **spring-core** en su versión **5.0.0.RELEASE**, pero nos encontramos con un conflicto de dependencias con otra biblioteca que utiliza **spring-core** en una versión diferente. Para resolver este conflicto y utilizar la versión deseada de spring-core, podemos agregar la siguiente dependencia en el archivo pom.xml:

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.0.0.RELEASE</version>
    <exclusions>
      <exclusion>
        <groupId>org.springframework</groupId>
        <artifactId>spring-beans</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
</dependencies>
```

Imagen 6: Elaboración propia.

En este ejemplo, se agrega la dependencia de spring-core en su versión 5.0.0.RELEASE. Además, se especifica la exclusión de la dependencia spring-beans de org.springframework, lo que significa que se excluirá cualquier versión de spring-beans que sea dependencia de spring-core. De esta manera, Maven utilizará la versión de spring-core que hemos especificado y evitará conflictos con la dependencia spring-beans que pueda existir en el proyecto.

Plugins

Los plugins son herramientas que se utilizan para extender la funcionalidad del sistema de construcción. Los plugins de Maven proporcionan tareas adicionales que pueden ser ejecutadas durante la construcción del proyecto, como la compilación del código fuente, la ejecución de pruebas unitarias, la generación de informes, la creación de paquetes de distribución, la publicación de artefactos en un repositorio, entre otras.

Los plugins de Maven son declarados en el archivo **pom.xml** del proyecto, en la sección **<build><plugins>**. Cada plugin tiene un conjunto de metas (goals) que se pueden ejecutar durante la construcción del proyecto. Por ejemplo, el plugin de compilación de Java tiene las metas compile, testCompile, jar, entre otras.

Maven proporciona una gran cantidad de plugins por defecto que cubren una amplia variedad de tareas de construcción, pero también es posible crear plugins personalizados para cubrir necesidades específicas de un proyecto. Además, los plugins de Maven pueden tener configuraciones adicionales, como propiedades y perfiles, que permiten personalizar su comportamiento y adaptarlo a las necesidades del proyecto.

Uno de los plugins más utilizados en Maven es el plugin ***maven-compiler-plugin***, que se utiliza para compilar el código fuente de un proyecto. Este plugin se incluye por defecto en la mayoría de los proyectos de Maven y es responsable de compilar el código fuente de Java en archivos de clase; y el plugin ***maven-surefire-plugin*** para la ejecución de pruebas unitarias. Este plugin se encarga de ejecutar las pruebas unitarias del proyecto y generar informes de los resultados.

Arquetipo (archetype)

En Maven, un arquetipo **es una plantilla o modelo de proyecto que sirve como punto de partida para crear nuevos proyectos con una estructura y configuración predefinida.** Un arquetipo incluye un conjunto de archivos y directorios que definen la estructura básica de un proyecto, así como la configuración necesaria para compilar, ejecutar y empaquetar el proyecto.

Los arquetipos permiten a los desarrolladores comenzar a trabajar en un nuevo proyecto con una estructura predefinida y una configuración inicial, lo que ahorra tiempo y esfuerzo en la creación manual de una estructura de proyecto y configuración. Además, los arquetipos pueden ser personalizados para adaptarse a las necesidades específicas de un proyecto.

Maven proporciona una variedad de arquetipos predefinidos para proyectos Java, como arquetipos para proyectos web, proyectos de consola, proyectos de biblioteca, entre otros. También es posible crear arquetipos personalizados para satisfacer las necesidades específicas de un proyecto.

Para crear un nuevo proyecto a partir de un arquetipo, se utiliza el comando "**mvn archetype:generate**" en la línea de comandos de Maven, proporcionando la información necesaria como el groupId, el artifactId, la versión y el arquetipo a utilizar.

Aquí se ofrece una lista de algunos arquetipos y cuál es su finalidad:

- **maven-archetype-quickstart**: Este es el arquetipo más básico y se utiliza para crear proyectos Java simples que no requieren ninguna configuración adicional. Este arquetipo crea una estructura de proyecto básica con un archivo pom.xml, una clase Main y un archivo de prueba.
- **maven-archetype-webapp**: Este arquetipo se utiliza para crear proyectos web que utilizan tecnologías como Servlets, JSP y JSTL. Este arquetipo crea una estructura de proyecto web básica con directorios para el código fuente, los recursos y las páginas JSP.
- **maven-archetype-quickstart-jdk8**: Este arquetipo es similar al maven-archetype-quickstart, pero está diseñado para proyectos que utilizan Java 8 o superior.
- **maven-archetype-j2ee-simple**: Este arquetipo se utiliza para crear proyectos J2EE simples que incluyen archivos de configuración para despliegue en servidores de aplicaciones.
- **maven-archetype-jar**: Este arquetipo se utiliza para crear proyectos de biblioteca que se empaquetan en archivos JAR.

- **maven-archetype-site**: Este arquetipo se utiliza para crear proyectos de documentación que utilizan Apache Maven Site Plugin para generar documentación en formato HTML.

Además de estos arquetipos, Maven ofrece muchos otros arquetipos para una variedad de proyectos y tecnologías, como proyectos de Spring, proyectos de Struts, proyectos de Hibernate, entre otros.

- **spring-boot-starter**: Este arquetipo se utiliza para crear aplicaciones Spring Boot con una configuración básica.
- **spring-mvc-webapp**: Este arquetipo se utiliza para crear aplicaciones web Spring MVC.
- **spring-webmvc-archetype**: Este arquetipo se utiliza para crear aplicaciones web Spring MVC basadas en XML.
- **spring-webmvc-quickstart-archetype**: Este arquetipo se utiliza para crear aplicaciones web Spring MVC con una configuración básica.
- **spring-hibernate-archetype**: Este arquetipo se utiliza para crear proyectos que utilizan Hibernate con Spring.

Conclusión

Maven es una de las herramientas de gestión de proyectos para Java mas usadas del mundo; todo desarrollador debe conocer y manejar esta herramienta. Dicho esto, se recomienda hacer el curso oficial de Maven (https://maven.apache.org/guides/getting-started/index.html)

BIBLIOGRAFÍA

Página oficial de Maven (https://maven.apache.org/)