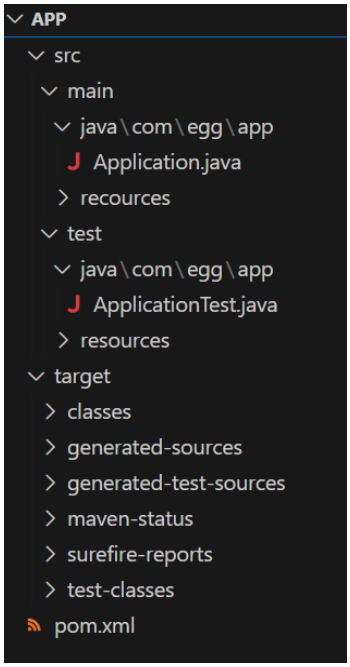
Introducción a Maven

Maven es una herramienta de construcción de proyectos y gestión de dependencias, ampliamente utilizada en la comunidad Java. Está diseñada para manejar el ciclo de vida completo de un proyecto de software, desde la etapa de compilación hasta las de prueba, empaquetado, despliegue y más allá.

Estructura proyecto Maven

Cuando creas un proyecto con Maven, este debe seguir una estructura de directorios establecida:



Como puedes observar, ahora dispones de una carpeta con el nombre de la aplicación, dentro de la cual se encuentra el archivo `pom.xml` de configuración, junto con subdirectorios para organizar los archivos. A continuación, te explico para qué sirve cada uno de ellos:

- **pom.xml**: Este es el archivo de configuración principal de un proyecto Maven. Contiene información relevante sobre el proyecto, incluidas las dependencias, los plugins de Maven utilizados, los perfiles de construcción, entre otros detalles (más adelante se ofrecerá más información sobre esto).

- **src/main/java/com/my-company/app/**: Este directorio alberga todos los archivos de código fuente de la aplicación. (Los nombres `my-company` y `app` deben ser reemplazados por los nombres correspondientes; en breve se dará más detalle sobre esto).
- **src/main/resources**: Este es el directorio donde se colocan todos los recursos de la aplicación, como archivos de configuración y propiedades (por ahora no lo utilizaremos).
- **src/test/java/com/my-company/app/**: Este directorio alberga todos los archivos de código fuente para pruebas. Estas pruebas unitarias se ejecutan para verificar la lógica en tus clases.
- **src/test/resources**: Este es el directorio donde se colocan los recursos de prueba, como los archivos de configuración para pruebas (aquí podrías colocar tus archivos CSV, por ejemplo).
- **target/**: Este directorio es donde Maven coloca archivos que se generan automáticamente, como los archivos de compilación del proyecto u otros que se crean mediante plugins.
- **target/classes**: Este es el directorio donde Maven coloca los archivos `.class` compilados cuando se construye el proyecto.

Archivo pom.xml

Como mencionamos anteriormente, el archivo `pom.xml` es donde se incluye toda la información necesaria para que Maven funcione bajo el enfoque de *Convención sobre Configuración*.

Convención sobre configuración

La *Convención sobre Configuración* es un enfoque para la configuración de proyectos que promueve la simplicidad, la consistencia y la estandarización mediante el uso de convenciones predefinidas.

En lugar de exigir una configuración exhaustiva y personalizada para cada aspecto de un proyecto, Maven establece convenciones y estructuras por defecto que deben seguirse. Esto elimina la necesidad de especificar configuraciones detalladas, permitiendo que los desarrolladores se concentren en la lógica de la aplicación en lugar de en la configuración. Este enfoque tiene varios beneficios:

👌 **Menos código de configuración**: Maven utiliza un conjunto de **convenciones predeterminadas**, lo que reduce significativamente la cantidad de configuración manual necesaria. Esto permite centrarse en el desarrollo sin preocuparse por detalles repetitivos de configuración

👌 **Estructura y configuración consistentes**: Todos los proyectos en Maven siguen un **formato estandarizado**, lo que facilita la comprensión y la colaboración en equipos. Gracias a esta uniformidad, cualquier desarrollador familiarizado con Maven puede integrarse rápidamente en un nuevo proyecto.

👌 **Mayor interoperabilidad**: Las convenciones de Maven aseguran que los proyectos sean **compatibles con diversas herramientas y sistemas externos**, simplificando la integración con frameworks, servidores y entornos de desarrollo.

👌 **Mayor productividad:** Al automatizar tareas repetitivas como la gestión de dependencias y la compilación del código, Maven permite a los desarrolladores **enfocarse en la lógica de la aplicación**, optimizando el tiempo y los recursos del equipo.

Estructura archivo pom.xml

El archivo `pom.xml` (Project Object Model) es el núcleo de cualquier proyecto Maven. Define la configuración, dependencias y reglas de construcción del proyecto. A continuación, se muestra un archivo `pom.xml` básico para una aplicación Java sin dependencias:

```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.mycompany</groupId>
    <artifactId>app</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
    <java.version>20</java.version>
        <maven.compiler.source>${java.version}</maven.compiler.source>
        <maven.compiler.target>${java.version}</maven.compiler.target>
```

```xml
    </properties>

    <dependencies>
        <!-- Aquí se agregan las dependencias del proyecto -->
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <source>${maven.compiler.source}</source>
                    <target>${maven.compiler.target}</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Conceptos clave del `pom.xml`

El `pom.xml` está basado en **XML (eXtensible Markup Language)**, un lenguaje de marcado que permite estructurar datos de forma legible tanto para humanos como para máquinas. A diferencia de **HTML**, que tiene etiquetas predefinidas para la presentación de contenido, **XML permite definir etiquetas personalizadas** según la estructura del documento.

Algunas etiquetas esenciales en `pom.xml` son:

- **`<project>:`** Es la etiqueta raíz del archivo. Contiene toda la configuración del proyecto.
- **`<modelVersion>:`** Define la versión del modelo POM utilizada. En todos los proyectos modernos de Maven, este valor es **4.0.0**, que se introdujo con Maven 2.0 en 2005.
- **`<groupId>:`** Identifica el grupo u organización del proyecto. Se recomienda seguir la convención de nombres de dominio invertido, como `com.ejemplo` o `org.miempresa`.

- **`<artifactId>:`** Es el nombre único del proyecto dentro del `groupId`. Este identificador se usará para generar el nombre del archivo `.jar` o `.war`.
- **`<version>:`** Indica la versión del proyecto. Al utilizar sistemas de control de versiones como **Git**, se recomienda actualizar este valor en cada versión publicada.
- **`<properties>:`** Define variables reutilizables dentro del `pom.xml`. En este caso, `java.version` permite especificar la versión de Java de forma centralizada.
- **`<dependencies>:`** Contiene la lista de dependencias necesarias para el proyecto. Cada dependencia se define con `<groupId>`, `<artifactId>` y `<version>`.
- **`<build>:`**Especifica configuraciones de compilación y construcción del proyecto.
- **`<plugins>:`**Define los plugins que Maven usará durante la construcción del proyecto.
- **`<plugin>:`**Especifica un plugin específico con sus configuraciones. En el ejemplo, el **`maven-compiler-plugin`** se usa para definir la versión de Java con la que se compilará el código.

Notas importantes

👉 En la mayoría de los proyectos Maven, las etiquetas `<project>`, `<modelVersion>`, `<groupId>`, `<artifactId>` y `<version>` son estándar y no suelen modificarse con frecuencia.

👉 Al definir propiedades en `<properties>`, se pueden reutilizar con la sintaxis `${nombre.de.la.propiedad}`, facilitando la gestión de configuraciones.

👉 Las dependencias se administran centralmente en `<dependencies>`, lo que simplifica la integración con bibliotecas externas sin necesidad de descargar archivos manualmente.

Dependencias en Maven

Una **dependencia** en Maven es una biblioteca o proyecto externo que tu aplicación necesita para compilar, probar o ejecutar correctamente. En lugar de descargar

manualmente los archivos JAR, Maven se encarga de gestionar y descargar automáticamente estas dependencias.

Esto es especialmente útil cuando compartes tu proyecto con otros programadores, ya que no es necesario enviar las bibliotecas junto con el código. Basta con que los colaboradores ejecuten el proyecto, y Maven descargará e integrará automáticamente todas las dependencias necesarias.

Ejemplo de una dependencia

Si necesitas agregar **JUnit 5** (una biblioteca para realizar pruebas unitarias en Java), solo debes incluir la siguiente dependencia en tu `pom.xml`:

```xml
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter</artifactId>
        <version>5.9.3</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

En este ejemplo, se especifican tres valores clave:

- **`<groupId>`**: Identifica la organización o proveedor de la biblioteca.
- **`<artifactId>`**: Nombre de la biblioteca.
- **`<version>`**: Especifica la versión que deseas utilizar.
- **`<scope>`**: Indica en qué fase del ciclo de vida del proyecto se requiere la dependencia (en este caso, solo para pruebas).

Dependencias transitivas

Maven también maneja automáticamente las **dependencias transitivas**, es decir, las dependencias de tus propias dependencias.

Por ejemplo:

- Tu proyecto **A** depende de la biblioteca **B**.
- La biblioteca **B** depende de la biblioteca **C**.
- Maven descargará **B** y **C** automáticamente, ya que **C** es una dependencia transitiva de **A**.

Gracias a este sistema, no es necesario declarar manualmente todas las dependencias secundarias, ya que Maven las administra por ti.

¿De dónde se descargan las dependencias?

Por defecto, Maven busca las dependencias en el **repositorio central de Maven**, un repositorio público donde muchos proyectos de código abierto publican sus bibliotecas.

Sin embargo, también puedes configurar repositorios adicionales si necesitas dependencias que no estén en el repositorio central.

¿Dónde se almacenan las dependencias descargadas?

Las dependencias **no se guardan en el directorio del proyecto**, sino en un repositorio local de Maven ubicado en tu sistema de archivos.

Por defecto, este repositorio se encuentra en:

- **Windows**: `C:\Users\TuUsuario\.m2\repository`
- **Linux/macOS**: `/home/tuusuario/.m2/repository`

Al utilizar este enfoque:

✔ **No necesitas descargar las mismas dependencias múltiples veces** para distintos proyectos.
✔ **Tus proyectos se mantienen más ligeros y organizados**, ya que las bibliotecas no se almacenan dentro del proyecto.

Alcances (`scope`) de las dependencias

La etiqueta **`<scope>`** define en qué etapas del ciclo de vida de Maven se necesita una dependencia. Los valores más comunes son:

Alcance (`scope`)	Descripción
compile (por defecto)	La dependencia está disponible en todas las fases del ciclo de vida. Se incluye en el código compilado y empaquetado.
test	La dependencia solo se usa en la fase de pruebas (no en la compilación normal ni en la ejecución).
runtime	No es necesaria para compilar, pero sí para ejecutar la aplicación.

provided	Necesaria para compilar, pero se espera que el entorno (por ejemplo, un servidor de aplicaciones) la proporcione en tiempo de ejecución.
system	Similar a `provided`, pero debes especificar manualmente la ubicación del archivo JAR en tu sistema.

👉 **Nota:** No necesitas preocuparte por elegir el `scope` adecuado. En la mayoría de los casos, las dependencias vienen con un `scope` predeterminado y solo es necesario modificarlo si tu aplicación lo requiere.

Plugins, Goals, Ciclo de Vida y Comandos

Plugin: Un plugin en Maven es un conjunto de herramientas que amplía la funcionalidad del proceso de construcción, permitiendo tareas como la compilación de código, la ejecución de pruebas o la creación de documentación. Los plugins son generalmente escritos en Java, pero en este curso nos enfocaremos en los plugins preexistentes que usaremos en nuestros proyectos.

Goals: Un *goal* es una tarea específica realizada por un plugin. Cuando ejecutas un comando como `mvn compile`, le estás indicando a Maven que ejecute el *goal* de compilación del plugin correspondiente.

Ciclo de vida: Maven organiza el proceso de construcción en tres ciclos de vida principales: **default**, **clean** y **site**. Cada ciclo de vida consta de una serie de fases que deben ejecutarse en un orden específico.

1. Ciclo de Vida Default:

Este ciclo es el principal y se ejecuta por defecto si no se especifica otro. A continuación, las fases más importantes:

- **validate:** Verifica que el proyecto es correcto y tiene toda la información necesaria.

- **compile:** Compila el código fuente.
- **test:** Ejecuta pruebas unitarias.
- **package:** Empaqueta el proyecto en un archivo JAR, WAR, etc.
- **verify:** Realiza verificaciones adicionales de calidad en el paquete.
- **install:** Instala el paquete en el repositorio local.
- **deploy:** Despliega el paquete en un repositorio remoto.

Nota: Ejecutar una fase también ejecutará todas las fases anteriores. Por ejemplo, `mvn install` ejecutará `validate`, `compile`, `test`, `package`, `verify` e `install`.

2. Ciclo de Vida Clean:

Este ciclo se enfoca en limpiar el proyecto eliminando archivos generados previamente:

- **clean:** Elimina el directorio `target/`, donde Maven guarda los archivos de salida.

3. Ciclo de Vida Site:

Este ciclo genera documentación para tu proyecto, creando un sitio web con la documentación generada:

- **site:** Crea la documentación en un directorio `target/site`.

Comandos:
Los comandos de Maven se ejecutan en la línea de comandos en el directorio raíz del proyecto. Ejemplo de algunos comandos:

- `mvn compile`: Compila el código.
- `mvn test`: Ejecuta las pruebas.
- `mvn package`: Empaqueta el proyecto.
- `mvn clean compile`: Limpia y recompila el proyecto desde cero.

Interacción entre Dependencias, Plugins y Ciclo de Vida:

Las dependencias se utilizan en diferentes fases del ciclo de vida, y los plugins se encargan de ejecutar los goals en esas fases. Por ejemplo, puedes tener una dependencia de pruebas en el ciclo de vida `test`, mientras que un plugin de compilación se ejecuta en la fase `compile`.

Directorio Target: El directorio `target` es el lugar donde Maven coloca todos los archivos generados durante la construcción. Esto incluye archivos compilados (`.class`), archivos empaquetados (`.jar`, `.war`, `.ear`), y otros artefactos del proyecto.

- **JAR (Java ARchive):** Archivo ejecutable que puede ser ejecutado con el JRE (Java Runtime Environment).
- **WAR (Web Application ARchive):** Usado para aplicaciones web, desplegables en servidores de aplicaciones.
- **EAR (Enterprise ARchive):** Para aplicaciones empresariales Java distribuidas.

Por defecto, Maven genera un archivo JAR al empaquetar un proyecto, pero puedes cambiar el comportamiento mediante el elemento `<packaging>` en el archivo `pom.xml`:

```
<project>
  ...
    <groupId>com.mycompany</groupId>
    <artifactId>app</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>
  ...
</project>
```

Arquetipos

Un *arquetipo* es un modelo o patrón para crear nuevos proyectos con una estructura estándar. Maven viene con varios arquetipos predefinidos, y puedes crear los tuyos propios. Los arquetipos son útiles para iniciar nuevos proyectos con la configuración base adecuada.

Para profundizar en el uso de Maven y explorar más detalles sobre su funcionamiento, te recomiendo que consultes la documentación oficial. En ella encontrarás información actualizada sobre los comandos, ciclos de vida, plugins, dependencias, y mucho más. Puedes acceder a la documentación en el siguiente enlace: **Documentación oficial de Apache Maven.**