Guía Práctica: Introducción a Apache Maven

Apache Maven es una herramienta de automatización de la construcción (*build*) y gestión de proyectos, basada en el concepto de configuración por convención. Está diseñada principalmente para proyectos en Java y permite gestionar dependencias, compilar código, ejecutar pruebas y generar artefactos para su distribución, todo mediante un solo archivo de configuración (`pom.xml`). Gracias a su sistema de repositorios, Maven simplifica la incorporación de bibliotecas externas y garantiza la coherencia en los proyectos a gran escala.

Objetivo

Esta práctica te permitirá familiarizarte con **Apache Maven**, una herramienta de gestión y construcción de proyectos Java. Aprenderás a configurar un proyecto, gestionar dependencias, construir artefactos, ejecutar pruebas con JUnit y empaquetar tu aplicación para su distribución.

Requisitos Previos

- Conocimiento de **Java** y fundamentos de desarrollo de software.
- Familiaridad con **Git** y **GitHub**.
- Instalación de **Java 17+** y **Maven**.
- Uso de un editor de código como **IntelliJ IDEA**, **Eclipse** o **VS Code** (*Opcional pero recomendable*).

Instalación de Maven

Antes de comenzar, asegúrate de tener **Apache Maven** instalado en tu sistema. A continuación, se presentan las instrucciones para instalar Maven en los principales sistemas operativos:

Windows

1. Descarga Maven desde la página oficial: https://maven.apache.org/download.cgi.

2. Extrae el contenido del archivo en una ubicación de tu elección (por ejemplo, `C:\Program Files\Apache\Maven`).

3. Configura la variable de entorno **MAVEN_HOME**:

   - Abre el **Panel de control** y dirígete a **Sistema > Configuración avanzada del sistema**.
   - En la pestaña **Opciones avanzadas**, haz clic en **Variables de entorno**.
   - Agrega una nueva variable con el nombre `MAVEN_HOME` y el valor de la ruta de Maven (por ejemplo, `C:\Program Files\Apache\Maven`).

4. Agrega Maven al **PATH**:

   - Edita la variable de entorno `Path` y agrega `%MAVEN_HOME%\bin`.

5. Verifica la instalación ejecutando:

   `mvn -version`

Ubuntu (Debian-based)

```
sudo apt update
sudo apt install maven -y
```

Para verificar la instalación:

`mvn -version`

Mac OS (usando Homebrew)

Si utilizas Homebrew, la instalación de Maven es sencilla:

`brew install maven`

Verifica que se haya instalado correctamente con:

`mvn -version`

---

1. Creación de un Proyecto con Maven

¿Qué es un archetype en Maven?

Un **archetype** es una plantilla predefinida que simplifica la creación de proyectos con una estructura estándar. Maven proporciona archetypes para distintos tipos de aplicaciones, como bibliotecas, aplicaciones web y microservicios.

Ejecuta el siguiente comando para generar un proyecto Java básico:

```
mvn archetype:generate -DgroupId=com.example \
  -DartifactId=demo \
  -DarchetypeArtifactId=maven-archetype-quickstart \
  -DinteractiveMode=false
```

Estructura generada:

```
demo/
├── src/
│   ├── main/java/com/example/
│   │   ├── App.java
│   ├── test/java/com/example/
│       ├── AppTest.java
├── pom.xml
```

Código de `App.java`

El comando mvn anterior genera el siguiente código de ejemplo para el proyecto:

```
package com.example;

public class App {
    public static void main(String[] args) {
        System.out.println("Hola, Maven!");
    }
}
```

---

2. Configuración del `pom.xml`

¿Qué define el `pom.xml`?

El archivo `pom.xml` centraliza la configuración del proyecto Maven, incluyendo:

- **Identificación** del proyecto (`groupId`, `artifactId`, `version`).
- **Dependencias** y sus versiones.
- **Plugins** para la automatización de tareas como compilación y empaquetado.
- **Perfiles** para personalizar la construcción.
- **Propiedades** para definir valores reutilizables.

¿Qué es un artifact en Maven?

Un **artifact** es un componente generado a partir del código fuente, como una biblioteca (`.jar`), una aplicación web (`.war`) o un ejecutable. Los artifacts permiten reutilizar código en distintos proyectos y facilitan la distribución del software. Cada artifact se identifica mediante un `groupId`, un `artifactId` y una `version`, lo que garantiza un control eficiente de las dependencias.

¿Cómo se gestionan las dependencias en `pom.xml`?

Las dependencias se definen en la sección `<dependencies>`, donde se especifica cada una con una sección `<dependency>`. Por ejemplo, para agregar JUnit como dependencia:

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
</dependency>
```

Manejo de Conflictos de versiones

Cuando un proyecto Maven utiliza múltiples dependencias, es posible que algunas de ellas requieran diferentes versiones de una misma biblioteca. Maven resuelve automáticamente estos conflictos utilizando la regla de **mayor profundidad** (más cercana en el árbol de dependencias), pero en algunos casos puede ser necesario forzar una versión específica.

El bloque `<dependencyManagement>` en `pom.xml` permite definir versiones específicas de dependencias para garantizar la coherencia en el proyecto.

Ejemplo:

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>5.3.20</version>
        </dependency>
    </dependencies>
</dependencyManagement>
```

Para excluir dependencias no deseadas:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <version>2.7.3</version>
    <exclusions>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

Análisis de Dependencias con Maven

Para mostrar el **árbol de dependencias**:

```
mvn dependency:tree
```

Ejemplo de salida:

```
...
[INFO] com.example:demo:jar:1.0-SNAPSHOT
[INFO] \- junit:junit:jar:4.13.2:test
[INFO]    \- org.hamcrest:hamcrest-core:jar:1.3:test
...
```

Para resolver dependencias:

```
mvn dependency:resolve
```

Ejemplo de salida:

```
...
[INFO] The following files have been resolved:
[INFO]    junit:junit:jar:4.13.2:test -- module junit [auto]
[INFO]    org.hamcrest:hamcrest-core:jar:1.3:test -- module hamcrest.core (a
[INFO]
...
```

---

3. Construcción del Proyecto

Maven maneja un ciclo de vida que incluye varias fases clave:

- **validate**: Verifica que el proyecto esté correctamente configurado.
- **compile**: Compila el código fuente.
- **test**: Ejecuta pruebas unitarias.
- **package**: Genera el artefacto (`.jar` o `.war`).
- **install**: Instala el artefacto en el repositorio local.
- **deploy**: Publica el artefacto en un repositorio remoto.

Para compilar el código fuente, usa:

```
mvn compile
```

Tras compilar, es posible que se genere un *warning* indicando que no se ha definido el *encoding* del archivo y que se está usando la codificación por defecto de la plataforma (por ejemplo, *UTF-8*).

La construcción de un artefacto es dependiente de la plataforma. Así que se recomienda incluir la siguiente configuración del Encoding UTF-8 en el archivo `pom.xml` para evitar problemas con caracteres especiales y asegurar que el proyecto use UTF-8:

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncodi
</properties>
```

Esto garantizará que tanto la compilación como la generación de informes usen la codificación adecuada.

---

4. Pruebas con JUnit

¿Qué es JUnit?

JUnit es un framework de pruebas unitarias para Java que permite verificar el comportamiento de los métodos en desarrollo.

Código de `AppTest.java`

El comando mvn inicial genera un archivo de prueba `AppTest.java` siguiente:

```
package com.example;

import static org.junit.Assert.assertTrue;
import org.junit.Test;

public class AppTest {
    @Test
    public void testApp() {
        assertTrue(true);
    }
}
```

Para ejecutar las pruebas:

```
mvn test
```

---

5. Empaquetado del Proyecto

¿Qué implica empaquetar un proyecto?

El empaquetado genera un **artifact** (`.jar` o `.war`) con el código compilado y sus dependencias. Esto permite distribuir el software y facilitar su reutilización.

Para empaquetar el proyecto, usa:

```
mvn package
```

El archivo resultante estará en `target/demo-1.0-SNAPSHOT.jar`.

Para ejecutarlo:

```
java -jar target/demo-1.0-SNAPSHOT.jar
```

¿Por qué tiene ese nombre el artefacto generado?

En Maven, el sufijo SNAPSHOT indica que el artefacto es una versión en desarrollo y aún no es una versión final estable. Esto permite a Maven gestionar actualizaciones automáticas de la versión más reciente del artefacto dentro de un mismo ciclo de desarrollo, sin necesidad de cambiar manualmente el número de versión. Cuando el proyecto está listo para una versión estable, se elimina el sufijo SNAPSHOT y se publica con un número de versión fijo, por ejemplo, 1.0.0.

---

6. Instalación y Despliegue

¿Para qué sirve `mvn install`?

El comando `mvn install` copia el `.jar` en el repositorio local de Maven (`~/.m2/repository`), permitiendo que otros proyectos lo usen como dependencia.

```
mvn install
```

¿Cómo desplegar un artifact en un repositorio remoto?

Para compartir un artifact en un repositorio remoto, usa:

```
mvn deploy
```

Esto requiere configurar credenciales en `settings.xml` y usar herramientas como **Nexus** o **Artifactory**.

---

7. Plugins y Perfiles en Maven

¿Qué son los plugins en Maven?

Los **plugins** amplían las capacidades de Maven, permitiendo realizar tareas adicionales como análisis de código o generación de reportes.

Un ejemplo de plugin ampliamente utilizado en Spring Boot es spring-boot-maven-plugin, que facilita la ejecución y empaquetado de aplicaciones Spring Boot:

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

Este plugin permite ejecutar una aplicación Spring Boot con el comando:

`mvn spring-boot:run`

También facilita la creación de un `.jar` ejecutable que incluye todas las dependencias necesarias para el despliegue.

¿Qué son los perfiles en Maven?

Los **perfiles** permiten definir configuraciones específicas según el entorno de ejecución (desarrollo, producción, pruebas).

```
<profiles>
    <profile>
        <id>produccion</id>
        <properties>
            <env>produccion</env>
        </properties>
    </profile>
</profiles>
```

Para activarlo:

```
mvn package -Pproduccion
```

Ejercicio Entregable

Crea un proyecto Maven con la solución al ejercicio propuesto en las clases de teoría **CreditCardTest**.

En este ejercicio se debe emplear el framework jUnit con el objetivo de apreciar su diseño y hacer un buen uso del framework desde una clase cliente externa al mismo. El ejercicio debe ser similar al estudiado en clase de teoría con el ejemplo de `ShoppingCart`.

Se pide:

- Codificar en Java una clase `CreditCard` que simule la interfaz y el comportamiento de una tarjeta de crédito. Podéis inspiraros en Github copilot, Codeium.io, ChatGPT, Gemini o herramientas similares para proponer una versión de dicha clase.
- Diseñar y codificar una suite de casos de prueba unitaria de nombre CreditCardTest para la clase CreditCard creada anteriormente. Usar jUnit 4 y/o jUnit 3 para codificarla.

El ejercicio se debe alojar en un repositorio github creado a tal efecto e indicar la URL del repositorio en la entrega.

Usad el archivo `README.md` en formato markdown del repositorio para incluir cualquier explicación y/o diagrama de diseño ilustrativo de la solución al ejercicio. Para confeccionar diagramas UML e incluirlos en el fichero .md podéis usar Mermaid o PlantUML.
