Taller UD08_01: Librerias Maven vs Jar

1. Introducción

En el desarrollo de software, es común que nuestros proyectos requieran funcionalidades específicas que no están incluidas en el lenguaje de programación o en las bibliotecas estándar. Por ejemplo, podríamos necesitar trabajar con archivos JSON, conectarnos a una base de datos, o realizar operaciones matemáticas complejas. En lugar de reinventar la rueda y escribir todo el código desde cero, podemos aprovechar bibliotecas externas que ya han sido desarrolladas y probadas por la comunidad.

Estas bibliotecas nos permiten ahorrar tiempo, reducir errores y enfocarnos en la lógica específica de nuestra aplicación. Sin embargo, para utilizar estas bibliotecas, es necesario agregarlas a nuestro proyecto. Aquí es donde entra en juego la decisión de si añadir manualmente un archivo JAR o utilizar una herramienta de gestión de dependencias como Maven.

Ambas opciones tienen sus ventajas y desventajas, y entender cuándo y cómo usarlas es fundamental para un desarrollo eficiente y mantenible.

2. Añadir un `JAR` Manualmente:

Cuando añades un JAR manualmente, estás descargando el archivo JAR de la biblioteca que necesitas y lo agregas directamente a tu proyecto. Esto implica que tú eres responsable de gestionar las dependencias, incluyendo la descarga de versiones compatibles y la resolución de conflictos entre bibliotecas.

- **Proceso en IntelliJ IDEA:**
    1. Descargas el JAR de la biblioteca que necesitas.
    2. En IntelliJ, vas a `File > Project Structure > Libraries`.
    3. Haces clic en el botón `+` y seleccionas "Java".
    4. Navegas hasta la ubicación del JAR y lo agregas.
    5. Asegúrate de que el JAR esté en el classpath de tu proyecto.
- **Ventajas:**
    - Control total sobre las bibliotecas que se utilizan.
- **Desventajas:**
    - Gestión manual de dependencias.
    - Dificultad para mantener actualizadas las bibliotecas.
    - Posibles conflictos de versiones.

2.1. Usar `Maven`:

Maven es una herramienta de gestión y comprensión de proyectos, principalmente utilizada en proyectos Java. Su principal función es gestionar las dependencias (librerías y otros componentes que tu proyecto necesita), construir el proyecto y gestionar la configuración del ciclo de vida del desarrollo. Maven utiliza un archivo de configuración denominado `pom.xml` (Project Object Model), donde se especifican las dependencias y otros detalles del proyecto.

- **Proceso en IntelliJ IDEA:**
    1. Creas un proyecto Maven en IntelliJ.
    2. En el archivo `pom.xml`, defines las dependencias que necesitas.
    3. IntelliJ automáticamente descarga las dependencias y las añade al classpath del proyecto.
- **Ventajas:**
    - **Gestión Automática de Dependencias:** Maven descarga automáticamente las dependencias del proyecto desde repositorios remotos y las incluye en el proyecto, evitando la necesidad de descargar y agregar manualmente los archivos JAR.
    - **Estandarización del Proyecto:** Proporciona una estructura estándar para los proyectos, lo que facilita la organización y el mantenimiento.
    - **Reproducibilidad:** El uso de `pom.xml` permite que cualquier desarrollador que clone el repositorio tenga exactamente las mismas versiones de las dependencias, asegurando que el proyecto se construya de manera consistente en diferentes entornos.
    - **Integración con IDEs:** Herramientas como IntelliJ IDEA tienen soporte nativo para Maven, lo que facilita la configuración y gestión del proyecto dentro del IDE.
    - **Gestión del Ciclo de Vida del Proyecto:** Maven puede automatizar tareas como compilación, pruebas, empaquetado y despliegue, facilitando la integración continua.
- **Desventajas:**
    - Curva de aprendizaje inicial.
    - Dependencia de la disponibilidad de los repositorios.

3. Comparativa

1. Gestión de Dependencias: en **Maven** las dependencias se especifican en el archivo `pom.xml`. Maven descarga las dependencias automáticamente desde los repositorios configurados. Mientras que en **Manual** las dependencias se descargan manualmente y se agregan al proyecto, lo que puede llevar a inconsistencias y errores si diferentes desarrolladores utilizan versiones distintas de las librerías.

2. Actualización de Dependencias: en **Maven** actualizar una dependencia es tan simple como cambiar la versión en el `pom.xml`. Maven se encarga de descargar la nueva versión. Por contra, en **Manual** requiere descargar la nueva versión manualmente y reemplazar los archivos JAR antiguos en el proyecto.

3. Estructura del Proyecto: en **Maven** proporciona una estructura de proyecto estandarizada, lo que facilita la comprensión y la navegación del proyecto. Mientras que en **Manual** la estructura del proyecto puede variar según el desarrollador, lo que puede llevar a confusión y desorganización.

4. Repositorios de Dependencias: en **Maven** utiliza repositorios remotos (como Maven Central) para buscar y descargar dependencias. También permite configurar repositorios internos. Pero en **Manual** las dependencias deben ser gestionadas y almacenadas manualmente, lo que puede ser engorroso y propenso a errores.

5. Integración con Herramientas de Construcción: Maven se integra fácilmente con herramientas de construcción y CI/CD, permitiendo la automatización de tareas como compilación, pruebas y despliegue. Por otro lado en **Manual** se requieren scripts personalizados y configuración adicional para integrar tareas de construcción y despliegue.

4. Actividades

4.1. Ejercicio 1: Añadir un JAR Manualmente

Debes descargar el JAR de una biblioteca común, como `Gson` (para manejo de JSON), y añadirlo manualmente a un proyecto en IntelliJ.

1. **Pasos a seguir:**
    - Descargar el JAR de Gson desde aquí.
    - Crear un nuevo proyecto en IntelliJ.
    - Añadir el JAR de Gson al proyecto como se describió anteriormente.
    - Escribir un programa simple que convierta un objeto Java a JSON usando Gson.
2. **Código de Ejemplo:**

```java
import com.google.gson.Gson;

class Persona {
    String nombre;
    int edad;

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
}

public class Main {
    public static void main(String[] args) {
        Gson gson = new Gson();
        String json = gson.toJson(new Persona("David", 35));
        System.out.println(json);
    }
}
```

4.2. Ejercicio 2: Usar Maven

Ahora debes crear un proyecto `Maven` en IntelliJ y añadir la dependencia de `Gson` a través del `pom.xml`.

1. **Pasos:**
    - Crear un nuevo proyecto Maven en IntelliJ.
    - Añadir la dependencia de Gson en el `pom.xml`.
    - Escribir el mismo programa que convierte un objeto Java a JSON usando Gson.
2. **Código de Ejemplo en `pom.xml`:**

```xml
<dependencies>
    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.10.1</version>
    </dependency>
</dependencies>
```

3. **Código de Ejemplo en Java:**

```java
import com.google.gson.Gson;

class Persona {
    String nombre;
    int edad;

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
```

```
        this.edad = edad;
    }
}

public class Main {
    public static void main(String[] args) {
        Gson gson = new Gson();
        String json = gson.toJson(new Persona("David", 35));
        System.out.println(json);
    }
}
```

5. Actividades

Genera dos proyectos, uno con JAR y otro con MAVEN (Ejercicio1 y Ejercicio2), cambia el nombre de la persona y pon el tuyo, y también tu edad. Genera un zip con los dos proyectos de IntelliJ. Envía el archivo zip a la tarea de Aules.

6. Fuentes de información

- Wikipedia
- https://www.deepseek.com/
- https://chatgpt.com/