Compilación de proyectos y gestión de dependencias en Java y Android

Marcos Núñez Celeiro – Aula Nosa 2020

Objetivos del curso

1) Comprender como funciona la compilación y ejecución de un proyecto Java (realizando manualmente tareas básicas que el IDE hace por nosotros).
2) Conocer las herramientas de manejo de proyectos existentes en varios lenguajes.
3) Aprender a manejar de forma básica Maven y Gradle.
4) Utilizar bibliotecas de terceros mediante Maven y Gradle.
5) Aprender a realizar una interfaz gráfica en Android de forma básica.

Instalación y herramientas del JDK y JRE

Compilación de un proyecto Java

- JDK: las siglas de Java Development Kit. Contiene el conjunto de herramientas necesarias para poder crear programas con código hecho en Java.
- JRE: siglas de “Java Runtime Environment”. Contiene las herramientas necesarias para poder ejecutar código Java en la JVM (la máquina virtual de Java).

Cuando se descarga el JDK viene incluido el JRE. Esto es porque si deseas realizar programas en Java, normalmente también quieres probarlos (es decir, ejecutarlos) y para esto necesitas el JRE.

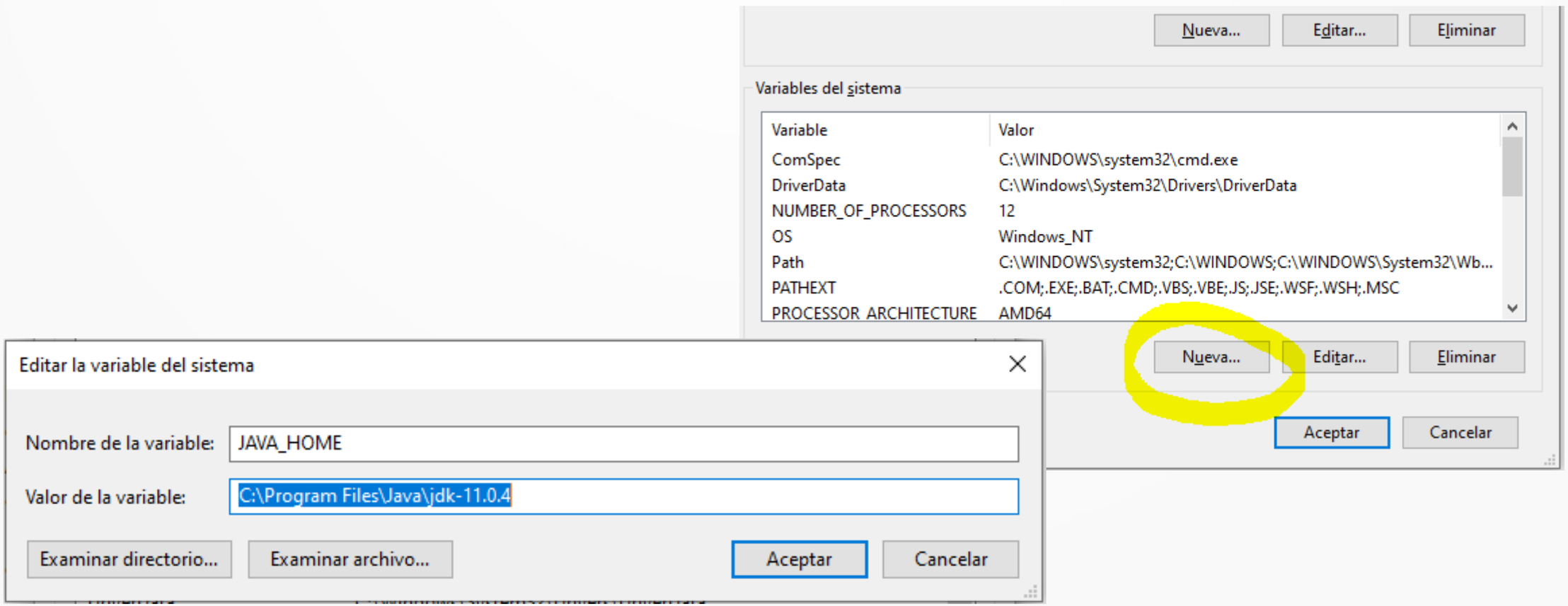
Herramientas contenidas en el JDK

Algunas herramientas que contiene el JDK son:

- **Javac**: compila código Java. Cuando en un IDE (por ejemplo Eclipse) guardas un fichero, se ejecuta la compilación.
- **Javap:** desensambla los ficheros .class.
- **Javadoc:** genera documentación en formato HTML del código.
- **Keytool:** permite crear claves y certificados para las aplicaciones.
- **Jarsigner:** permite firmar ficheros .jar.
- **Jshell:** consola REPL, permite ejecutar código Java desde línea de comandos (existe desde Java 9).

Ejercicio: añade el JDK al PATH

- Para comprobar si el JDK está añadido al PATH puedes intentar ejecutar directamente alguno de los comandos anteriores (p. ej: javac).
- Si no lo está crea una variable **JAVA_HOME** y concaténala a la variable **PATH**.



Ejercicio: compila un proyecto

- Crea una clase HelloWorld.
- Compila la clase (se creará el .class).
- Ejecuta el programa.
- Envuelve el programa en un JAR.

Javac <clase>

java <fichero.class>

jar -cvf <NombreDelJar.jar> <paquete con los .class o clase>

Gestión automática de proyectos y sus dependencias

Gestión automática de dependencias

- Existen herramientas utilizadas en casi cualquier lenguaje de programación para configurar y compilar proyectos hechos en ese lenguaje así como automatizar la descarga y configuración de bibliotecas.
- Ejemplos donde son útiles estas herramientas:
  - Necesitas descargar Bootstrap para utilizar en tu página web.
  - Necesitas añadir el conector de MySQL para conectarte a una base de datos en Java o C#
  - Necesitas descargar bibliotecas que utilizan, a su vez, otras bibliotecas. Para una sola cosa descargas 5 ficheros .jar diferentes a mano y los copias en tu proyecto.

Herramientas de manejo de dependencias

- Maven: la más utilizada con Java, utiliza XML para la configuración. Su fichero es el **pom.xml**.
- Gradle: más moderna que Maven, más potente y más compleja. Su fichero de configuración es el **build.gradle**.
- Ant: más antigua y con un 10% de uso aproximadamente. Su fichero de configuración es **build.xml**.
- NPM: gestor de paquetes de NodeJS. Guarda las descargas en una carpeta llamada "node_modules". Su fichero de configuración es el **package.json**.
- YARN: permite manejar dependencias en la vista (en el *frontend*).
- SBT: para Scala. Su fichero de configuración es
- Composer: permite manejar dependencias facilmente en PHP.

Maven

- Herramienta para la gestión y construcción de proyectos Java.
- Creada en 2002.
- Es muy antigua pero muy estable.
- Permite: compilar proyectos, automatizar distintas tareas, ejecutar tests y descargar automaticamente bibliotecas.

Ejercicio: creación de un proyecto

Crea un proyecto en Java con los siguientes ficheros (no hace falta que metas contenido dentro):

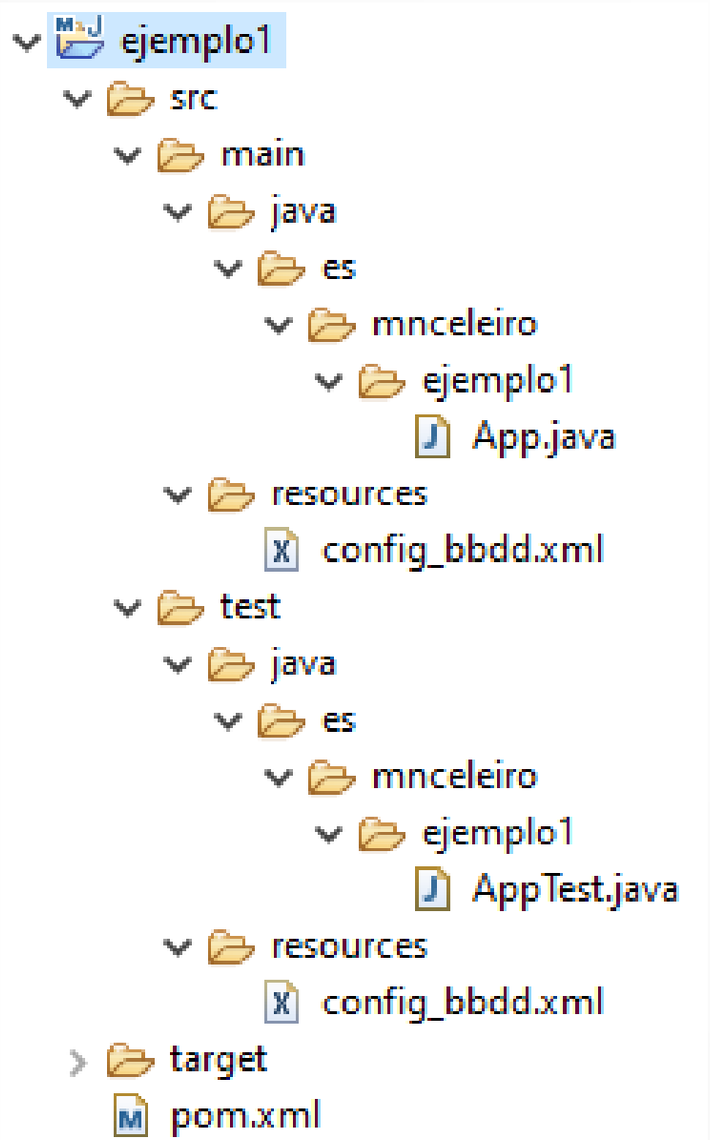
- usuarios.xml
- peliculas.xml
- ScriptBaseDeDatos.sql
- HolaMundo.java
- persona.json
- Un fichero para tests unitarios (de JUnit).

Organización de ficheros

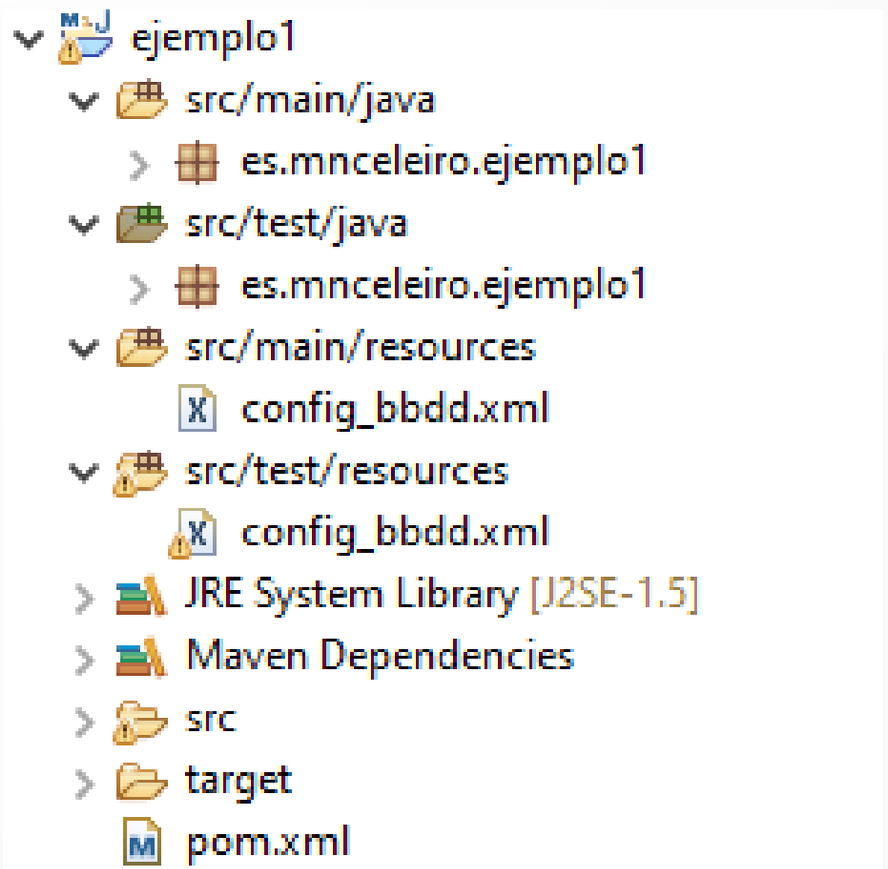
- Si se ha realizado el ejercicio anterior probablemente todos hayáis organizado el proyecto de manera diferente.
- Maven proporciona una estructura de carpetas estándar con la que se crearán sus proyectos Java (de esta manera, todo el mundo la comprende y trabajan sobre lo mismo).
  - **src (source code, código fuente):** contiene el código de la aplicación y los ficheros de recursos (.xml, .json, .properties, etc.). Esta carpeta contiene las siguientes:
    - **main:** código y recursos de la aplicación.
    - **test:** código y recursos de los tests (p. ej: en JUnit).

    Las carpetas MAIN y TEST pueden contener las tres siguientes carpetas:
    - **java:** clases java que contienen el código fuente de la aplicación
    - **resources:** recursos que se incluirán en el empaquetado y serán usados por la aplicación, como pueden ser ficheros de texto o scripts
    - **webapp:** ficheros que tienen que ver con la vista (HTML, JSP, Javascript, CSS, etc.).
  - **Fichero pom.xml:** es el único fichero propio y obligatorio de MAVEN. Aquí se define todo lo que queremos hacer.

Organización de ficheros



¿Cuál de estas dos maneras de organizar un proyecto se corresponde con la utilizada por MAVEN?



Ventajas y desventajas

- Ventajas
  - Creación de proyectos organizados en carpetas y con código de manera muy rápida.
  - Manejo fácil de dependencias (bibliotecas).
  - Descarga automática de dependencias.
  - Existen *plugins* hechos por otras personas para diversas tareas: gestión de servidores web, tests, etc.).
- Desventajas
  - XML se empieza a considerar un poco “desfasado”.
  - Otras herramientas, como Gradle, son más potentes (aunque a veces esa potencia puede ser innecesaria).

Ejercicio: instalación de Maven

- Accede a https://maven.apache.org para descargar la última versión.
- Descomprime el fichero descargado y colócala donde consideres adecuado. Un sitio podría ser el siguiente:
  - C:\Aplicaciones\apache-maven-3.6.3.
- Añade Maven al PATH.
- Prueba que esté correctamente añadido con el comando:
  - *mvn --version*.

Conceptos básicos y definiciones

Configuración

- El fichero de configuración de Maven es el pom.xml.

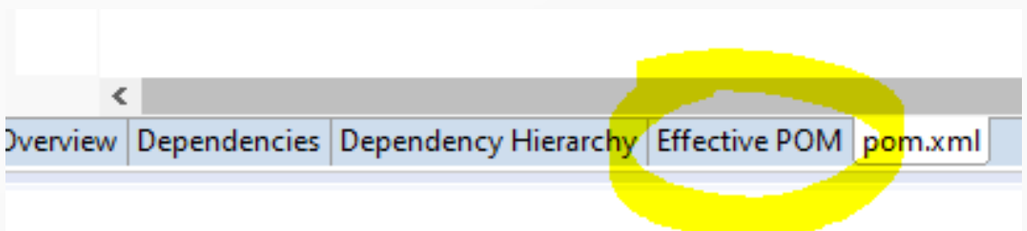
- **Blanco: cabeceras.**
- **Verde: coordenadas.**
- **Negro: variables.**
- **Marrón: bibliotecas o dependencias.**

```
<?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
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>es.ejemplo1</groupId>
  <artifactId>hello-world</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <java.version>11</java.version>
    <maven.compiler.source>${java.version}</maven.compiler.source>
    <maven.compiler.target>${java.version}</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>TEXTO</groupId>
      <artifactId>TEXTO</artifactId>
      <version>TEXTO</version>
    </dependency>
  </dependencies>
</project>
```

Fichero POM.XML

- Los ficheros pom.xml pueden tener padres e hijos. Esto es, tu pom.xml puede heredar de otro pom.xml y así sucesivamente.
- El pom.xml describe cómo se comporta el proyecto Maven (qué dependencias tiene, cuál es su artifactId, versión, etc.).
- Existe un pom.xml por defecto que tiene una serie de cosas definidas (y con lo que tu pones en tu pom.xml lo que haces es modificar ese pom.xml por defecto. La combinación de ambos se llama **effective pom** o pom efectivo.

Comando: mvn help:effective-pom



Maven coordinates (coordenadas)

- Se denominan así los siguientes elementos del fichero de configuración **pom.xml**:
  - **groupId**: normalmente la URL de la compañía al revés (por ejemplo: es.google). Suele coincidir con el nombre del paquete principal de la aplicación.
  - **artifactId**: suele coincidir con el nombre del proyecto.
  - **version**: versión actual del proyecto.
- La idea es que **estas coordenadas sean únicas y no haya ningún proyecto con las mismas tres coordenadas** (de esta manera, con esas tres se puede identificar de manera unívoca un aplicación o biblioteca).

Dependencias

- Las dependencias, librerías o bibliotecas son basicamente proyectos de código de los que depende el tuyo. Por ejemplo:
  - Si creas un nuevo proyecto con unas clases y métodos y lo exportas a un JAR. Posteriormente lo importas desde Eclipse en otro proyecto y usas sus clases. En este caso el primer proyecto que has hecho es una librería o dependencia que usa el segundo.

Arquetipos

- Un arquetipo en Maven es una plantilla para tus carpetas y código. Puedes crear un proyecto nuevo basado en un arquetipo de manera que se creen automáticamente todos los directorios, paquetes, ficheros de configuración y clases que desees.
- Existen múltiples arquetipos por defecto en Maven, pero además cualquiera pueda crearse sus propios arquetipos (p. ej: podrías tener alguno para proyectos nuevos en tu empresa con una configuración por defecto).

Repositorios

- Existe un repositorio "**central**" (Maven central). Se encuentra en la siguiente URL: https://repo1.maven.org/maven2/
- En el enlace anterior se pueden ver las bibliotecas disponibles para descarga, aunque es poco intuitivo para buscar.
  - En https://mvnrepository.com/ se puede encontrar de forma sencilla cualquier biblioteca.
- Además de este repositorio central se dispone de un repositorio "**local**". Este está en nuestro ordenador (en la carpeta .m2 que está oculta) y aquí se almacenarán todas las bibliotecas descargadas de cualquier proyecto.
- Por último, las empresas o usuarios pueden tener **repositorios creados por ellos** (públicos o privados).

Ejercicio I: arquetipos

- Revisa la documentación oficial de Maven y fíjate en como se organizan los siguientes arquetipos:
  - Simple.
  - Quickstart.
  - Webapp.

- La documentación está en: https://maven.apache.org/archetypes/

Ejercicio I: arquetipos

- En la parte de abajo de la página de cada arquetipo aparece un comando para generar un proyecto a partir de un arquetipo (mediante el comando **mvn archetype:generate**). Escoge uno de esos tres arquetipos anteriores y crea un nuevo proyecto utilizando este comando.
- Crea un nuevo proyecto a partir de uno de los tres arquetipos (que no sea el del ejercicio anterior), **pero ahora hazlo utilizando un IDE**.
- Revisa los pom.xml y directorios de los dos proyectos que has creado.

Maven goals

- **Validate**: verifica que el proyecto es correcto.
- **Compile**: invoca el compilador de Java y convierte el código fuente en bytecode (básicamente compila el proyecto por nosotros).
- **Test**: ejecuta los tests que tengamos.
- **Package**: busca los ficheros compilados (.class) y los los mueve a un jar, war o ear (dependiendo de lo que se ponga en el pom.xml).
- **Install**: instala el paquete en tu repositorio local (.m2) para poder utilizarlo en otros proyectos si así lo deseas.
- **deploy**: copia el repositorio local a remoto para compartirlo con otros desarrolladores.

Estas fases se ejecutan en orden. Por ejemplo: si se ejecuta el comando "**mvn install**" **se ejecutará** no solo "install", sino **todo lo anterior también ordenadamente**.

Ejercicio: creación de proyecto con Eclipse

En el proyecto que has creado:

- Busca en **mvnrepository** la biblioteca **commons-lang3** de **Apache**.
- Añade la biblioteca a tu proyecto.
- Utiliza alguna clase de esta biblioteca. Si quieres puedes intentarlo con el método ***capitalize*** de ***StringUtils***

Ejercicio: crea una biblioteca

- Crea un pequeño modelo de datos con Maven y utilízalo como dependencia desde el proyecto anterior mediante Maven.
- Crea un proyecto de *gradle* y añade una dependencia.

Repaso

- Responde a las siguientes preguntas. Si no sabes alguna puedes utilizar internet.
- ¿Que significan las siglas POM?
- ¿Qué es y para qué sirve un fichero XSD?
- Que término se usa para denominar la combinación de: groupId, artifactId y versión?
- Si quieres saber qué versión de Maven tienes, ¿que comando debes ejecutar?
- ¿Que comando te permite crear un jar de tu proyecto Maven?
- En qué directorio sitúa Maven los ficheros que se producen al compilar?
- ¿Cómo añades una biblioteca de otra persona u organización a tu proyecto Maven?
- ¿En qué directorio se almacenan todas las bibliotecas que descargas con Maven?

Referencias

- Documentación oficial de Maven: https://maven.apache.org/
- Maven: beginner to guru. https://www.udemy.com/course/apache-maven-beginner-to-guru
- Ejecución de un JAR: https://www.programmergate.com/run-jar-file-command-line/
- Librería commons-lang: https://commons.apache.org/proper/commons-lang/
- Ejecución de un JAR desde CLI: https://www.programmergate.com/run-jar-file-command-line/
- https://docs.oracle.com/javase/tutorial/deployment/jar/downman.html
- http://eljaviador.com/comprender-maven-parte-02.html
- https://stackoverflow.com/questions/41202120/what-does-it-mean-that-artifact-scope-is-not-transitive
- https://www.mkyong.com/jdbc/connect-to-oracle-db-via-jdbc-driver-java/
- Documentación oficial de Java 11: https://docs.oracle.com/en/java/javase/11/tools/tools-and-command-reference.html