Fue desarrollado por Apache y tiene por objetivo simplificar la construcción de proyectos y la gestión de dependencias.
Tiene una estructura predefinida y poco modificable en comparación con Gradle.
La configuración del proyecto está basada en XML y está almacenada en POM (Project Object Model) que es un archivo xml que contiene la información y los detalles de configuración utilizados por maven para construir el proyecto.

El archivo **POM** está estructurado por:
GroupID: Identifica de manera única al proyecto a través de su grupo de organización.
ArtifactID: Es el ID único para el proyecto dentro del grupo.
Version: Especifica la versión actual del proyecto.
Dependencies: Define las dependencias externas requeridas para el proyecto.
Build: Contiene la configuración relacionada con el proceso de construcción del proyecto.
Plugins: Lista los complementos de Maven utilizados para ampliar o modificar el proceso de construcción.

```xml
<?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>org.example</groupId>
    <artifactId>Sprint_1_IT_ACADEMY</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.11</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.hamcrest</groupId>
            <artifactId>hamcrest</artifactId>
            <version>2.2</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.assertj</groupId>
            <artifactId>assertj-core</artifactId>
            <version>3.24.2</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter</artifactId>
            <version>RELEASE</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>7.1.0</version>
```

Maven
Gradle

Al copiar groupID, artifactId y version de un proyecto y lo pegamos como dependencia en otro proyecto. Podemos utilizar las clases y funcionalidades proporcionadas por ese proyecto.
Maven se encarga de buscar esa dependencia en el repositorio de Maven central o definidos en tu configuración Maven.

La estructura de un proyecto Maven (src, target, pom.xml).

SRC -> Contenido para código fuente y recursos
TARGET -> Donde se almacenan los resultados de la construcción.
POM.XML -> La configuración del proyecto. (Project Object Model)

Comandos más utilizados en Maven.

Clean ->Borra todos los .class y .jar generados.
Compile -> Compila el código fuente del proyecto.
Test -> Ejecuta las pruebas unitarias.
Package -> Empaqueta el proyecto en un archivo JAR o WAR.
Install -> LLeva el jar a nuestro repositorio local de jars. Queda "visible" para otros proyectos maven en nuestro ordenador.
Deploy ->Lleva el jar a nuestro servidor de jars. Queda "visible" para otros proyectos maven en otros ordenadores. Este comando necesita que a maven se le haya indicado dónde está dicho servidor de jars.

Que es un JAR?
Es un Java ARchive, que es un tipo de formato que se utiliza para distribuir aplicaciones Java, bibliotecas y componentes.
Contienen archivos comprimidos, incluyendo archivos de clase, archivos de metadatos y recursos como imágenes y archivos de configuración.
Dentro está el MANIFEST.MF que es donde están los metadatos y recursos.

Permite que la aplicación se inicie simplemente ejecutando el archivo JAR.
Tanto Maven y Gradle se encargan de descargar automáticamente los JAR necesarios para repositorios remotos.

Es multi-lenguaje. JAVA,C + +,Python y más.
Tiene una sintaxis más legible y expresiva, lo que permite una configuración más flexible y concisa en comparación con Maven.
La pregunta más común al iniciar un proyecto con Gradle, Groovy o Kotlin?

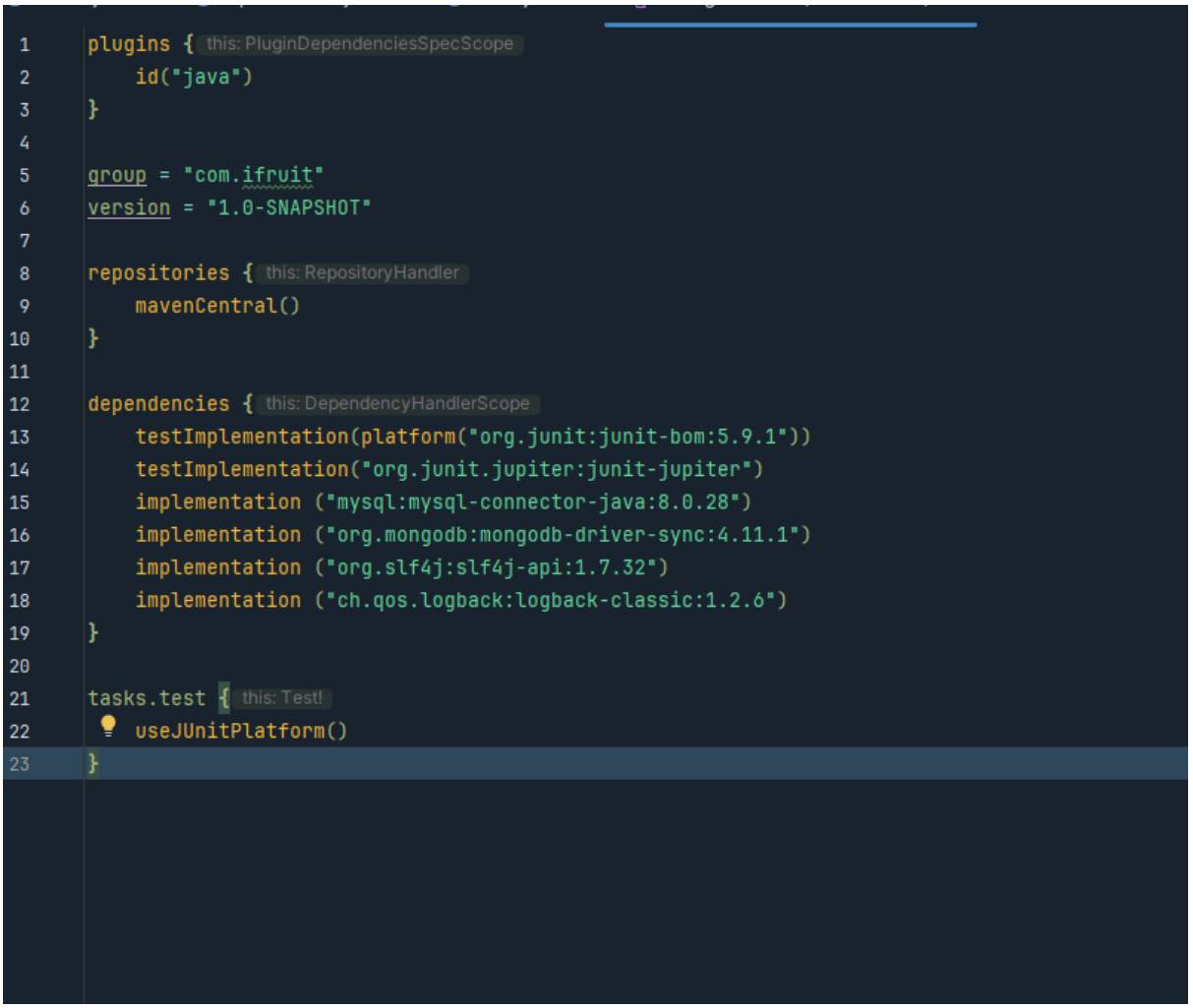
Groovy era el lenguaje de configuración predeterminado en Gradle y ha sido utilizado durante mucho tiempo.
La sintaxis de Groovy puede ser más familiar para los que tienen conocimientos previos en Java. Tiene un amplio soporte y una gran cantidad de recursos y bibliotecas disponibles.
.
Por otro lado, Kotlin es un lenguaje más moderno con funciones de extensión y sintaxis más concisa. Se está volviendo más popular que Groovy.
En sí no hay muchas más diferencias, dependerá de las preferencias del equipo y las características del proyecto.

Ventajas de Gradle:
Flexibilidad: Utiliza un DSL (Lenguaje Específico de Dominio) basado en Groovy para definir tareas de construcción. Esto hace que Gradle sea más flexible y personalizable que Maven.
Rendimiento: Utiliza un modelo de construcción incremental para acelerar la construcción de proyectos.
Soporte multi-proyecto que facilita la construcción de proyectos grandes y complejos.

Maven™
Gradle

Gradle genera un build.gradle o build.gradle.kts que está estructurado en:

Plugins: Define los complementos necesarios para el proyecto.
Repositories: Especifica los repositorios de donde se descargarán las dependencias del proyecto.
Dependencies: Lista las dependencias externas requeridas para el proyecto.
Tasks: Define tareas personalizadas o configuraciones especificas del proyecto.
Build: Contiene la configuración relacionada con el proceso de construcción del proyecto. Como la version java, compilación..empaquetado y distribución.



Los comandos más utilizados en Gradle són:
gradle build: Construye el proyecto.
gradle clean: Limpia el directorio de construcción.
gradle tasks: Muestra una lista de tareas disponibles en el proyecto.
gradle dependencies: Muestra las dependencias del proyecto.
Se pueden encadenar tantas tareas como quieras.
-> gradle clean compileJava

Maven™
Gradle

Diferencias entre Maven y Gradle:

En **Maven**, siempre tienes que escribir una serie de etiquetas
para realizar ciertas tareas,con **Gradle** permite lo mismo, con menos código.

Estructura de configuración: Maven utiliza XML **(pom.xml),** mientras que Gradle utiliza **Groovy** o **Kotlin.**

Flexibilidad: Gradle ofrece más flexibilidad y capacidad de personalización que Maven.
Rendimiento: Gradle tiende a ser más rápido en la construcción de proyectos debido a su modelo incremental.
Curva de aprendizaje: Maven es más fácil de empezar debido a su enfoque basado en convenciones, mientras que Gradle puede requerir una curva de aprendizaje más pronunciada debido a su flexibilidad.

La clave está en la flexibilidad, si el proyecto se puede gestionar con metas estándar o con poca personalización entonces **Maven** es suficiente, por otro lado si el proyecto requiere diferentes comportamientos en función de múltiples variables en tiempo de construcción quizá **Gradle** responda más eficazmente a estos requerimientos.