Introduction to Maven and Gradle

Overview of Build Automation Tools

Build automation tools help developers streamline the process of building, testing, and deploying software projects. They take care of repetitive tasks like compiling code, managing dependencies, and packaging applications, which makes development more efficient and error-free.

Two popular tools in the Java ecosystem are **Maven** and **Gradle**. Both are great for managing project builds and dependencies, but they have some key differences.

Maven

- **What is Maven?** Maven is a build automation tool primarily used for Java projects. It uses an XML configuration file called `pom.xml` (Project Object Model) to define project settings, dependencies, and build steps.
- **Main Features:**
  - •Predefined project structure and lifecycle phases.
  - •Automatic dependency management through Maven Central.
  - •Wide range of plugins for things like testing and deployment.
  - •Supports complex projects with multiple modules.

Gradle

- **What is Gradle?** Gradle is a more modern and versatile build tool that supports multiple programming languages, including Java, Groovy, and Kotlin. It uses a domain-specific language (DSL) for build scripts, written in Groovy or Kotlin.
- **Main Features:**
  - •Faster builds thanks to task caching and incremental builds.
  - •Flexible and customizable build scripts.
  - •Works with Maven repositories for dependency management.
  - •Excellent support for multi-module and cross-language projects.
  - •Integrates easily with CI/CD pipelines.

Key Differences Between Maven and Gradle

Aspect	Maven	Gradle
Configuration	XML (`pom.xml`)	Groovy or Kotlin DSL
Performance	Slower	Faster due to caching
Flexibility	Less flexible	Highly customizable
Learning Curve	Easier to pick up	Slightly steeper
Script Size	Verbose	More concise
Dependency Management	Uses Maven Central	Compatible with Maven too
Plugin Support	Large ecosystem	Extensible and versatile

Installation and Setup

How to Install Maven:

1. **Download Maven:**
   - •Go to the Maven Download Page and download the latest binary ZIP file.
2. **Extract the ZIP File:**
   - •Right-click the downloaded ZIP file and select **Extract All…** or use any extraction tool like WinRAR or 7-Zip.
3. **Move the Folder:**
   - •After extraction, move the extracted **Maven folder** (usually named **`apache-maven-x.x.x`**) to a convenient directory like `C:\Program Files\`.
4. **Navigate to the `bin` Folder:**
   - •Open the **Maven folder**, then navigate to the **`bin`** folder inside.
   - •Copy the path from the File Explorer address bar(e.g., **`C:\Program Files\apache-maven-x.x.x\bin`**).
5. **Set Environment Variables:**
   - •Open the **Start Menu**, search for **Environment Variables**, and select **Edit the system environment variables**.
   - •Click **Environment Variables**.

- Under **System Variables**:
  - Find the **path**, double click on it and click **New**.
  - Paste the full path to the `bin` folder of your Maven directory (e.g., **`C:\Program Files\apache-maven-x.x.x\bin`**).

6. **Save the Changes:**
   - Click **OK** to close the windows and save your changes.
7. **Verify the Installation:**
   - Open Command Prompt and run: **`mvn -v`** If Maven is correctly installed, it will display the version number.

How to install Gradle

1. **Download Gradle:**

Visit the Gradle Downloads Page and download the latest binary ZIP file.

2. **Extract the ZIP File:**
   - Right-click the downloaded ZIP file and select **Extract All…** or use any extraction tool like WinRAR or 7-Zip.
3. **Move the Folder:**
   - After extraction, move the extracted **Gradle folder** (usually named **`gradle-x.x.x`**) to a convenient directory like `C:\Program Files\`.
4. **Navigate to the `bin` Folder:**
   - Open the **Gradle folder**, then navigate to the **`bin`** folder inside.
   - Copy the path from the File Explorer address bar (e.g., **`C:\Program Files\gradle-x.x\bin`**).
5. **Set Environment Variables:**
   - Open the **Start Menu**, search for **Environment Variables**, and select **Edit the system environment variables**.
   - Click **Environment Variables**.
   - Under **System Variables**:
     - Find the **path**, double click on it and click **New**.
     - Paste the full path to the `bin` folder of your Gradle directory (e.g., **`C:\Program Files\gradle-x.x.x\bin`**).
6. **Save the Changes:**
   - Click **OK** to close the windows and save your changes.
7. **Verify the Installation:**
   - Open a terminal or Command Prompt and run: **`gradle -v`** If it shows the Gradle version, the setup is complete.

Program:2

1: Install the Java JDK

- If you haven't installed the **Java JDK** yet, you can follow the link below to download and install it. Download Java JDK from Oracle

Working with Maven is a key skill for managing Java-based projects, particularly in the areas of build automation, dependency management, and project configuration. Below is a guide on creating a Maven project, understanding the POM file, and using dependency management and plugins:

Overview of the Project

2: Creating a Maven Project

There are a few ways to create a Maven project, such as using the command line, IDEs like IntelliJ IDEA or Eclipse, or generating it via an archetype.

1. **Using Command Line**:
   - To create a basic Maven project using the command line, you can use the following command:

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

- **`groupId`:** A unique identifier for the group (usually the domain name).
- **`artifactId`:** A unique name for the project artifact (your project).
- **`archetypeArtifactId`:** The template you want to use for the project.
- **`DinteractiveMode=false`:** Disables prompts during project generation.

This will create a basic Maven project with the required directory structure and **`pom.xml`** file.

2. **Using IDEs**

Most modern IDEs (like IntelliJ IDEA or Eclipse) provide wizards to generate Maven projects. For example, in IntelliJ IDEA:

1. Go to **File > New Project**.
2. Choose **Maven** from the list of project types.
3. Provide the **`groupId`** and **`artifactId`** for your project.

3: Understanding the POM File

The **POM (Project Object Model)** file is the heart of a Maven project. It is an XML file that contains all the configuration details about the project. Below is an example of a simple POM file:

A basic `pom.xml` structure looks like this:

```
<?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>com.example</groupId>
        <artifactId>my-project</artifactId>
        <version>1.0-SNAPSHOT</version>
        <packaging>jar</packaging>

        <dependencies>
                <!-- Dependencies go here -->
        </dependencies>

        <build>
                <plugins>
                        <!-- Plugins go here -->
                </plugins>
        </build>
</project>
```

Key element in `pom.xml`:

- **`<groupId>`:** The group or organization that the project belongs to.
- **`<artifactId>`:** The name of the project or artifact.
- **`<version>`:** The version of the project (often follows a format like `1.0-SNAPSHOT`).
- **`<packaging>`:** Type of artifact, e.g., `jar`, `war`, `pom`, etc.
- **`<dependencies>`:** A list of dependencies the project requires.
- **`<build>`:** Specifies the build settings, such as plugins to use.

4: Dependency Management

Maven uses the `<dependencies>` tag in the `pom.xml` to manage external libraries or dependencies that your project needs. When Maven builds the project, it will automatically download these dependencies from a repository (like Maven Central).

Example of adding a dependency:

```
<dependencies>
        <dependency>
                <groupId>org.apache.commons</groupId>
                <artifactId>commons-lang3</artifactId>
                <version>3.12.0</version>
        </dependency>
</dependencies>
```

- **Transitive Dependencies**
  - Maven automatically resolves transitive dependencies. For example, if you add a library that depends on other libraries, Maven will also download those.
- **Scopes**
  - Dependencies can have different scopes that determine when they are available:
    - **compile** (default): Available in all build phases.
    - **provided**: Available during compilation but not at runtime (e.g., a web server container).
    - **runtime**: Needed only at runtime, not during compilation.
    - **test**: Required only for testing.

5: Using Plugins

Maven plugins are used to perform tasks during the build lifecycle, such as compiling code, running tests, packaging, and deploying. You can specify plugins within the `<build>` section of your **`pom.xml`**.

- **Adding Plugins**
  - You can add a plugin to your `pom.xml` like so:

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.1</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
    </plugins>
</build>
```

In this example, the **`maven-compiler-plugin`** is used to compile Java code and specify the source and target JDK versions.

1. **Common Plugins**
   - **maven-compiler-plugin**: Compiles Java code.
   - **maven-surefire-plugin**: Runs unit tests.
   - **maven-jar-plugin**: Packages the project as a JAR file.
   - **maven-clean-plugin**: Cleans up the `target/` directory.
2. **Plugin Goals** Each plugin consists of goals, which are specific tasks to be executed. For example:
   - **`mvn clean install`:** This will clean the target directory and then install the package in the local repository.
   - **`mvn compile`:** This will compile the source code.
   - **`mvn test`:** This will run unit tests.

6: Dependency Versions and Repositories

1. **Version Ranges**
   - You can specify a version range for dependencies, allowing Maven to choose a compatible version automatically. for example:

```
<dependency>
    <groupId>com.google.guava</groupId>
    <artifactId>guava</artifactId>
    <version>[30.0,)</version> <!-- Guava version 30.0 or higher -->
</dependency>
```

2. **Repositories**
   - Maven primarily fetches dependencies from Maven Central, but you can also specify custom repositories. For example:

```
<repositories>
```

```
    <repository>
        <id>custom-repo</id>
        <url>https://repo.example.com/maven2</url>
    </repository>
</repositories>
```

Working with Maven Project

Note: Always create separate folder to do any program**.**

- Open command prompt.
- **`mkdir program2`** – this will create **program2** folder.
- **`cd program2`** – navigate program2 folder.
- After then follow the below step to working with Maven project.

Step 1: CStep 4: Open Java Code (`AppTest.java`) File

- Open a file **`AppTest.java`** inside the **`src/test/java/com/example/`** directory.
- After opening the **AppTest.java** copy the below code and paste it in that file then save it.

reating a Maven Project

- You can create a **Maven project** using the **`mvn`** command (or through your **IDE**, as mentioned earlier). But here, I'll give you the essential **`pom.xml`** and **Java code**.
- Let's use the **Apache Commons Lang library** as a **dependency** (which provides utilities for **working with strings**, **numbers**, etc.). We will use this in a **simple Java program** to **work with strings**

Step 2: Open The `pom.xml` File

- You can manually navigate the **project folder** named call **myapp** and open the file pom.xml and copy the below code and paste it then save it.
- In case if you not getting project folder then type command in your cmd.
  - **cd myapp** – is use to navigate the project folder.
  - **notepad pom.xml** – is use to open pom file in notepad.

Step 3: Open Java Code (`App.java`) File

- Open a file **`App.java`** inside the **`src/main/java/com/example/`** directory.
- After opening the **App.java** copy the below code and paste it in that file then save it.

```
package com.example;

public class App {

    public int add(int a, int b) {
        return a + b;
    }

    public static void main(String[] args) {
        App app = new App();

        int result = app.add(2, 3);
        System.out.println("2 + 3 = " + result);

        System.out.println("Application executed successfully!");
    }
}
```

Step 4: Open Java Code (`AppTest.java`) File

- Open a file **`AppTest.java`** inside the **`src/test/java/com/example/`** directory.
- After opening the **AppTest.java** copy the below code and paste it in that file then save it.

Program:3

Gradle Project Overview

1: Setting Up a Gradle Project

- **Install Gradle** (If you haven't already):
  - Follow Gradle installation Program 1 **click here**
- **Create a new Gradle project**: You can set up a new Gradle project using the Gradle Wrapper or manually. Using the Gradle Wrapper is the preferred approach as it ensures your project will use the correct version of Gradle.
- **To create a new Gradle project using the command line:**

3: Dependency Management

Gradle provides a powerful dependency management system. You define your project's dependencies in the `dependencies` block.

1. **Adding dependencies**:
   - Gradle supports various dependency scopes such as `implementation`, `compileOnly`, `testImplementation`, and others.
2. **Declaring repositories**: To resolve dependencies, you need to specify repositories where Gradle should look for them. Typically, you'll use Maven Central or JCenter, but you can also configure private repositories.

4: Task Automation

Gradle tasks automate various tasks in your project lifecycle, like compiling code, running tests, and creating builds.

1. **Using predefined tasks**: Gradle provides many predefined tasks for common activities, such as:
   - **`build`** – compiles the project, runs tests, and creates the build output.
   - **`test`** – runs tests.
   - **`clean`** – deletes the build output.
2. Example of running in the build task
3. **Creating custom tasks**: You can define your own tasks to automate specific actions. For example, creating a custom task to print a message.

5: Running Gradle Tasks

To run a task, use the following command in the terminal:

For example:

- To run the `build` task: **`gradle build`**
- To run a custom task: **`gradle printMessage`**

6: Advanced Automation

You can define task dependencies and configure tasks to run in a specific order. Example of task dependency:

In this case, **`secondTask`** will depend on the completion of **`firstTask`** before it runs.

Working with Gradle Project (Groovy DSL):

Step 1: Create a new Project

```
gradle init --type java-application
```

- while creating project it will ask necessary requirement:
  - **Enter target Java version (min: 7, default: 21):** 17
  - **Project name (default: program3-groovy):** groovyProject
  - **Select application structure:**
    - 1: Single application project
    - 2: Application and library project
      - **Enter selection (default: Single application project) [1..2]** 1
  - **Select build script DSL:**
    - 1: Kotlin
    - 2: Groovy
      - **Enter selection (default: Kotlin) [1..2]**

- **Select test framework:**
  - 1: JUnit 4
  - 2: TestNG
  - 3: Spock
  - 4: JUnit Jupiter
    - **Enter selection (default: JUnit Jupiter) [1..4]** 1
- **Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no]**
  - no

Step 3: `AdditionOperation.java`(Change file name and update below code)

- After creating project **change the file name**.
- Manually navigate the folder path like **src/main/java/org/example/**
- Change the file name **App.java** to **AdditionOperation.java**
- After then open that file and copy the below code and past it, save it.

Step 4: `AdditionOperationTest.java` (JUnit Test) (Change file name and update below code)

- After creating project **change the file name**.
- Manually navigate the folder path like **src/test/java/org/example/**
- Change the file name **AppTest.java** to **`AdditionOperationTest`.java**
- After then open that file and copy the below code and past it, save it.

Step 5: Run Gradle Commands

- To **build** the project:

Working with Gradle Project (Kotlin DSL):

Step 1: Create a new Project

```
gradle init --type java-application
```

- while creating project it will ask necessary requirement:
  - **Enter target Java version (min: 7, default: 21):** 17
  - **Project name (default: program3-kotlin):** kotlinProject
  - **Select application structure:**
    - 1: Single application project
    - 2: Application and library project
      - **Enter selection (default: Single application project) [1..2]** 1
  - **Select build script DSL:**
    - 1: Kotlin
    - 2: Groovy
      - **Enter selection (default: Kotlin) [1..2]** 1
  - **Select test framework:**
    - 1: JUnit 4
    - 2: TestNG
    - 3: Spock
    - 4: JUnit Jupiter
      - **Enter selection (default: JUnit Jupiter) [1..4]** 1
  - **Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no]**
    - no

Step 3: `Main.kt` (Change file name and update below code)

- After creating project **change the file name**.
- Manually navigate the folder path like **src/main/java/org/example/**
- Change the file name **App.java** to **Main.kt**
- After then open that file and copy the below code and past it, save it.

Step 4: `MainTest.kt` (JUnit Test) (Change file name and update below code)

- After creating project **change the file name**.
- Manually navigate the folder path like **src/test/java/org/com/example/**

- Change the file name **MainTest.java** to **`MainTest.kt`**
- After then open that file and copy the below code and past it, save it.

Program:4

Step 1: Creating a Maven Project

You can create a **Maven project** using the **`mvn`** command (or through your **IDE**, as mentioned earlier). But here, I'll give you the essential **`pom.xml`** and **Java code**.

- **I'm Using Command Line:**
  - To create a basic Maven project using the command line, you can use the following command:

Step 2: Open The `pom.xml` File

- You can manually navigate the **project folder** named call **maven-example** and open the file pom.xml and copy the below code and paste it then save it.
- In case if you not getting project folder then type command in your cmd.
  - **cd maven-example** – is use to navigate the project folder.
  - **notepad pom.xml** – is use to open pom file in notepad.

Step 3: Open Java Code (`App.java`) File

- Open a file **`App.java`** inside the **`src/main/java/com/example/`** directory.
- After opening the **App.java** copy the below code and paste it in that file then save it.

Step 4: Run the Project

To build and run this project, follow these steps:

- Open the terminal in the project directory and run the following command to build the project.

Run the program with below command:

```
mvn exec:java -Dexec.mainClass="com.example.App"
```

Step 5: Migrate the Maven Project to Gradle

1. **Initialize Gradle**: Navigate to the project directory (**`gradle-example`**) and run:

```
gradle init
```

- It will ask **Found a Maven build. Generate a Gradle build from this? (default: yes) [yes, no]**
  - Type **Yes**
- **Select build script DSL:**
  - 1: Kotlin
  - 2: Groovy
  - Enter selection (default: Kotlin) [1..2]
    - Type **2**
- **Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no]**
  - Type **No**

Navigate the project folder and open **`build.gradle`** file then add the below code and save it.

Step 6: Run the Gradle Project

- **Build the Project**: In the project directory (`gradle-example`), run the below command to build the project:

Step 7: Verify the Migration

- **Compare the Output:** Make sure that both the **Maven** and **Gradle** builds produce the same output:
  - **Maven Output**:

```
Hello, Maven
This is the simple realworld example....
Sum of 5 and 10 is 15
```

- **Gradle Output**:

program :5

Introduction to Jenkins

What is Jenkins?

Jenkins is an open-source automation server widely used in the field of Continuous Integration (CI) and Continuous Delivery (CD). It allows developers to automate the building, testing, and deployment of software projects, making the development process more efficient and reliable.

Key features of Jenkins:

- **CI/CD**: Jenkins supports Continuous Integration and Continuous Deployment, allowing developers to integrate code changes frequently and automate the deployment of applications.
- **Plugins**: Jenkins has a vast library of plugins that can extend its capabilities. These plugins integrate Jenkins with version control systems (like Git), build tools (like Maven or Gradle), testing frameworks, deployment tools, and much more.
- **Pipeline as Code**: Jenkins allows the creation of pipelines using Groovy-based DSL scripts or YAML files, enabling version-controlled and repeatable pipelines.
- **Cross-platform**: Jenkins can run on various platforms such as Windows, Linux, macOS, and others.

Installing Jenkins

Jenkins can be installed on local machines, on a cloud environment, or even in containers. Here's how you can install Jenkins in Window local System environments:

1. Installing Jenkins Locally

Step-by-Step Guide (Window):

1. **Prerequisites:**
   - Ensure that J**ava (JDK) is installed** on your system. **Jenkins requires Java 21**. If not then **click here**.
   - You can check if Java is installed by running `java -version` in the terminal.
2. **Install Jenkins on Window System):**
   - Download the Jenkins Windows installer from the official Jenkins website.
   - Run the installer and follow the on-screen instructions. While installing choose **login system**: **run service as LocalSystem (not recommended)**.
   - After then use **default port** or you can **configure you own port like I'm using port 3030** then **click on test** and **next**.
   - After then **change the directory** and choose **java jdk-21** path look like **C:\Program Files\Java\jdk-21**.
   - After then **click next, next** and then it will **ask permission click on yes** and it will start installing.
   - After successfully installed, Jenkins will be **running on port either default port** or **chosen port** like i choose **port 3030** by default (you can access it in your browser at `http://localhost:8080`) or `http://localhost:3030`.

2. Jenkins Setup in browser:

- After opening browser by visiting your local address the browser should look like below screenshot.

It will ask **administrator password** so you have to **navigate the above highlighted path** and open that `initialAdminPassword` **in notepad** or any software to **see the password**.

- Just c**opy that password and paste** it and **click on continue**.
- It will ask to **customize Jenkins** so **click on install suggested plugin** it will **automatically install all required plugin**.
- After then **create admin profile** by **filling all details** then **click on save and continue** after then **save and finish** after then **click on start using Jenkin**.