**Experiment 1: Introduction to Maven and Gradle  
Objective:**To understand what **Maven** and **Gradle** are, compare their features, and learn how to install and configure them for Java project build automation. **Introduction to Maven and Gradle: Build automation tools** are used to simplify tasks such as compiling code, managing dependencies, running tests, and packaging software. They save time, reduce manual errors, and help developers streamline software development and deployment processes.Two widely used build tools in the Java ecosystem are **Maven** and **Gradle**. **Maven  
What is Maven:** Maven is a **build automation tool** primarily used for Java projects. It uses an XML-based configuration file called pom.xml (Project Object Model) to define dependencies, project structure, and build lifecycle phases. **Main Features: >**Predefined project structure and lifecycle. **>** Automatic dependency resolution through **Maven Central Repository**. **>** Rich plugin ecosystem for testing, deployment, etc. **>** Supports large, modular, and multi-module projects. **Gradle  
What is Gradle:** Gradle is a modern, powerful, and flexible build tool that supports not only Java but also Groovy, Kotlin, Scala, and more. Instead of XML, Gradle uses **domain-specific languages (DSL)** like Groovy or Kotlin for writing build scripts. **Main Features:>** Faster build performance using **incremental builds** and **task caching**. **>** Flexible and highly customizable via Groovy/Kotlin DSL. **>** Compatible with Maven repositories. **>** Integrates easily with **CI/CD pipelines**. **>** Well-suited for **multi-language and multi-module** projects. **Key Differences Between Maven and Gradle (Simplified)**

| **Aspect** | **Maven** | **Gradle** |
| --- | --- | --- |
| Configuration Format | XML (pom.xml) | Groovy or Kotlin (DSL) |
| Performance | Slower | Faster due to caching |
| Flexibility | Less flexible | Highly customizable |
| Learning Curve | Easier to learn | Slightly more complex |
| Script Size | Verbose | More concise |
| Dependency Management | Maven Central | Uses Maven repos; very flexible |
| Plugin Support | Large ecosystem | Extensible and versatile |

**Installation and Setup Instructions  
Part A: Installing Maven  
Step 1: Install Java (JDK)  
>** Open terminal. **>** Update system packages:sudo apt update **>** Install default OpenJDK:sudo apt install default-jdk -y  **>** Verify Java installation:java -version  
**Step 2: Download and Install Maven**Visit: <https://maven.apache.org/download.cgi>  
Copy the download link for the version (e.g., 3.9.9).  
Use wget to download: wget https://downloads.apache.org/maven/maven-3/3.9.9/binaries/apache-maven-3.9.9-bin.tar.gz -P /tmp  
Extract to /opt: sudo tar xf /tmp/apache-maven-3.9.9-bin.tar.gz -C /opt  
Create a symbolic link: sudo ln -s /opt/apache-maven-3.9.9 /opt/maven  
**Step 3: Set Environment Variables**Create Maven profile script: sudo nano /etc/profile.d/maven.sh  
Paste the following: **>** export JAVA\_HOME=/usr/lib/jvm/default-java **>** export M2\_HOME=/opt/maven **>** export MAVEN\_HOME=/opt/maven **>** export PATH=${M2\_HOME}/bin:${PATH}  
Make it executable: sudo chmod +x /etc/profile.d/maven.sh  
Apply the settings: source /etc/profile.d/maven.sh  
**Step 4: Verify Maven Installation:** mvn -version  
**Part B: Installing Gradle  
Step 1: Install Java**Update system: sudo apt update  
Install default JDK: sudo apt install default-jdk -y  
Verify: java --version

**Step 2: Download Gradle**Visit: <https://gradle.org/releases>Copy the download link for the binary distribution (e.g., gradle-8.12-bin.zip).Use wget:wget <https://services.gradle.org/distributions/gradle-8.12-bin.zip> **Step 3: Extract and Configure**Extract Gradle:sudo unzip -d /opt/gradle gradle-8.12-bin.zipCreate the profile script:sudo nano /etc/profile.d/gradle.shPaste the following: **>** export GRADLE\_HOME=/opt/gradle/gradle-8.12 **>** export PATH=${GRADLE\_HOME}/bin:${PATH}Make it executable:sudo chmod +x /etc/profile.d/gradle.shApply:source /etc/profile.d/gradle.sh **Step 4: Verify Gradle Installation**gradle –version  
**Experiment 2: Working with Maven  
Objective: >** Learn how to create a Maven project. **>**Understand the structure and purpose of the pom.xml file. **>**Use dependencies and plugins. **>** Compile, test, and package the project. **Step 1: Creating a Maven Project**Run the following command to generate a new Maven project:mvn archetype:generate -DgroupId=com.example -DartifactId=myapp -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=falseAfter running the command, a folder named myapp will be created with the basic Maven structure. **Step 2: Editing the POM File (pom.xml)**Navigate into the project directory:cd myappOpen the pom.xml file in a text editor:nano pom.xmlReplace the contents with the following code:<?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>myapp</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- JUnit Dependency for Testing -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Maven Surefire Plugin for running tests -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.22.2</version>

<configuration>

<redirectTestOutputToFile>false</redirectTestOutputToFile>

<useSystemOut>true</useSystemOut>

</configuration>

</plugin>

</plugins>

</build>

</project>

**What is a pom.xml file:** It is the **Project Object Model** file that defines your project’s configuration.  
It manages: > Project metadata >Dependencies > Plugins > Build lifecycle and configurations  
**Step 3: Create and Edit Java Source File (App.java)**  
Navigate to the following directory: cd src/main/java/com/example/  
Open App.java: nano App.java  
Paste the following code:  
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: Create and Edit Test File (AppTest.java)**  
Navigate to: cd src/test/java/com/example/  
Open AppTest.java: nano AppTest.java  
Paste the following test code:  
package com.example;  
import org.junit.Assert;  
import org.junit.Test;  
public class AppTest {

@Test

public void testAdd() {

App app = new App();

int result = app.add(2, 3);

System.out.println("Running test: 2 + 3 = " + result);

Assert.assertEquals(5, result);

}

}

**Step 5: Build and Run the Maven Project**

**1. Compile the Project:** mvn compile **2. Run Unit Tests:** mvn test **3. Package into a JAR:** mvn package **4. Run the Application:** java -cp target/myapp-1.0-SNAPSHOT.jar com.example.App