Programming in Java

5. Gradle

Gradle

- Initial Release: 2007
  - Creators: Hans Dockter and the Gradleware (now Gradle Inc.) team
  - Motivation: To improve upon the limitations of
    - *Apache Ant (too low-level)*
    - *Maven (rigid, XML-heavy, and difficult to customize).*
  - Emerged to address the limitations of older tools like Ant and Maven
  - offering a scriptable, high-performance, and deeply extensible build system.
  - Used for all kinds of projects from Android apps to enterprise-scale JVM systems
  - Intended for both speed and flexibility in modern DevOps and CI/CD workflows.

Gradle vs Maven

Feature	Maven	Gradle
Build Script Format	Declarative XML (`pom.xml`)	Scripted Groovy (`build.gradle`)
Readability	Clear structure, verbose	Concise, but requires scripting knowledge
Build Logic Customization	Difficult and plugin-dependent	Easily scriptable and extendable
Performance	Slower (no build caching or incrementality)	Faster due to task caching and incremental builds
Dependency Management	Stable, convention-based, transitive resolution	Same as Maven, but more flexible and scriptable
Multi-module Projects	Clear conventions, but rigid structure	Flexible layout and dependency resolution
Plugin Support	Vast, mature ecosystem (mostly XML config)	Rich plugin ecosystem, with more dynamic behavior
Build Lifecycle	Fixed lifecycle (`validate` , `compile` , `test`)	Flexible task graph execution
IDE Support	Excellent (IDEA, Eclipse, NetBeans)	Excellent (especially IntelliJ IDEA and Eclipse)
Tooling Support	Stable, with mature integrations	Good, with additional Gradle Tooling API
Learning Curve	Easier for Java devs familiar with XML	Slightly steeper due to Groovy scripting

When Maven May Be the Better Choice

Feature	Maven Advantage
Ease of Use	XML configuration is **explicit and readable**—easier for beginners and teams that prefer convention over customization.
Predictability	Strict **lifecycle and project structure** result in highly predictable builds.
Convention over Flexibility	Favors **standardization** with limited customization, reducing complexity and errors.
Learning Curve	Easier to grasp for Java developers who already understand **XML and declarative builds**.
Documentation & Community	Long-standing ecosystem with **extensive documentation**, examples, and StackOverflow support.
Tool Compatibility	Seamless integration with most Java IDEs, CI/CD tools, and repositories like Nexus and Artifactory.
Mature Plugin Ecosystem	Thousands of stable plugins for code quality, testing, deployment, and reporting.
Multi-module Simplicity	Easy-to-manage multi-module builds with well-defined inheritance and dependency resolution via `parent` POMs.
Standardized Output	Default layout and build outputs are highly predictable (`target/`, `*.jar`, etc.).
No Scripting Knowledge Needed	Does not require Groovy or any scripting—ideal for teams that prefer configuration over programming.

Why Prefer Gradle Over Maven

Reason	Gradle Advantage
Faster Builds	Supports **incremental builds**, **task output caching**, and **parallel execution** for much better performance.
Flexible Build Logic	Build logic can be **customized using Groovy scripts**—no need for writing custom plugins just to do simple things.
Compact and Concise Syntax	`build.gradle` is **shorter and easier to write** than Maven's verbose XML `pom.xml`.
Dynamic Task Graph	Gradle computes tasks **lazily and conditionally**, executing only what is necessary.
Superior Multi-module Support	Gradle handles large **multi-project builds** more flexibly and efficiently.
Dependency Version Conflict Resolution	More flexible **dependency resolution strategies** compared to Maven's strict nearest-wins.
Built-in Wrapper (`gradlew`)	Ensures **consistent build environment** across machines without needing Gradle pre-installed.
Strong Android Support	Official build system for Android Studio with deep integration. Maven is not supported.
Rich Plugin Ecosystem	Many powerful plugins with **easier configuration and scripting control** than Maven plugins.
Modern Architecture & APIs	Designed for **extensibility, IDE integration, and performance**, with a modern Tooling API.
Better Support for Continuous Delivery	Gradle is optimized for **CI/CD pipelines**, supporting build scans, caching, and parallelism.

Build Comparison

- The two images on the right compare how the same project would be defined in Maven and Gradle

```xml
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>helloworld</artifactId>
  <version>1.0.0</version>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.13.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

```
plugins {
    id 'java'
}

group = 'com.example'
version = '1.0.0'

repositories {
    mavenCentral()
}

dependencies {
    testImplementation 'junit:junit:4.13.2'
}
```

How Gradle Works

- Gradle works based on these key ideas:
  - Project: Represents the thing you are building.
  - Tasks: Units of work like compile, test, jar, or run)
  - Build: A sequence of tasks.
  - Build Scripts: Define how the project is built, what plugins and dependencies it uses.
- When you run a build (gradle build)
  - Gradle loads the project and reads build.gradle
  - Creates a task graph based on what you asked it to do.
  - Executes the tasks in order, respecting dependencies.

How Gradle Works

- Gradle works based on these key ideas:
  - Project: Represents the thing you are building.
  - Tasks: Units of work like compile, test, jar, or run)
  - Build: A sequence of tasks.
  - Build Scripts: Define how the project is built, what plugins and dependencies it uses.
- When you run a build (gradle build)
  - Gradle loads the project and reads build.gradle
  - Creates a task graph based on what you asked it to do.
  - Executes the tasks in order, respecting dependencies.

Typical Gradle Project Structure

```
my-app/
├── build.gradle          # Main build script
├── settings.gradle       # Project name and module setup
├── gradle/               # Wrapper scripts
│   └── wrapper/
├── gradlew               # Unix wrapper
├── gradlew.bat           # Windows wrapper
└── src/
    ├── main/
    │   └── java/
    │       └── App.java
    └── test/
        └── java/
            └── AppTest.java
```

Anatomy of a build.gradle File

- Key Sections:
  - plugins – Applies the Java and Application plugins.
  - group and version – Project coordinates (like Maven's groupId and version).
  - repositories – Where to find dependencies.
  - dependencies – What external libraries to use.
  - application – Defines the main class to run the app.

```
plugins {
    id 'java'
    id 'application'
}

group = 'com.example'
version = '1.0.0'

repositories {
    mavenCentral()
}

dependencies {
    implementation 'com.google.guava:guava:32.0.0'
    testImplementation 'junit:junit:4.13.2'
}

application {
    mainClassName = 'com.example.App'
}
```

Adding Dependencies

- Gradle handles dependencies through configurations like:
  - Implementation: Used at compile and runtime
  - TestImplementation: Used for test code only
- Gradle downloads dependencies from Maven Central or other configured repositories.

```
dependencies {
    implementation 'org.apache.commons:commons-lang3:3.12.0'
}
```

Adding Dependencies

- Gradle handles dependencies through configurations like:
  - Implementation: Used at compile and runtime
  - TestImplementation: Used for test code only
- Gradle downloads dependencies from Maven Central or other configured repositories.

```
dependencies {
    implementation 'org.apache.commons:commons-lang3:3.12.0'
}
```

Common Gradle Commands

- gradle build: Compiles code and runs tests
- gradle run: Runs the application
- gradle test: Runs unit tests
- gradle clean: Deletes the build/ directory
- gradle dependencies: Shows resolved dependency tree
- ./gradlew:
  - Runs Gradle via the wrapper (recommended)

Using ./gradlew (Gradle Wrapper)

- Rrecommended tool for running Gradle builds. I
  - A script that launches Gradle using a specific version declared by the project
- For commands `./gradlew build`
  - The will check if the required Gradle version is available
  - It will download it if necessary, then use that Gradle version to execute the build
  - You do not need a Gradle installation on your system to build the project
  - The wrapper handles it for you
- Reasons for using the script
  - Consistent Gradle version
  - No manual installation needed
  - Consistent build environment
  - Easy version upgrades

Using Plugins

- Plugins extend Gradle's functionality.
- Built-in examples:
  - java – Compiles Java source code.
  - application – Adds tasks to run a Java app.
  - checkstyle, jacoco, etc.

Lab 5-1

Building a Gradle Project


Java™