UNIT II COMPILE AND BUILD USING MAVEN & GRADLE 6

Introduction, Installation of Maven, POM files, Maven Build lifecycle, Build phases(compile build, test, package) Maven Profiles, Maven repositories(local, central, global),Maven plugins, Maven create and build Artifacts, Dependency management, Installation of Gradle, Understand build using Gradle

---

GRADLE

Gradle is an open source build automation tool used primarily for Java projects, that automates the creation of software build including the process of compile (converting to binary), test (executing automated process), package (bundling) and deploy (publishing it into shared, central repository).

Gradle can handle multiple languages — including Python, JavaScript, C/C++, and others through specific plugins and task definitions. It was launched in 2007 but released in 2009. Gradle overcomes the drawbacks of maven and Ant.

Gradle is written in Groovy language for its build scripts. Groovy is a powerful, dynamic, and scripting-friendly programming language for the Java platform. It's often called "Java's flexible cousin" because:

- It runs on the Java Virtual Machine (JVM).
- It looks and feels like Java, but is simpler and more expressive.
- It can use any Java library directly.

Drawback of Maven and Ant:

Drawback in Maven / Ant	How Gradle Overcomes It
Maven: Uses only XML for configuration and is hard to customize. **Ant**: No standard project structure, everything must be manually defined.	Gradle uses **Groovy or Kotlin DSL**, to write build logic like real code — more concise, easier to read, and fully customizable.

Maven: Build lifecycle is fixed and hard to modify. **Ant**: Too much flexibility — no conventions, which can lead to messy builds.	Gradle uses **"convention over configuration"** but allows **full customization** when needed — best of both worlds.
Maven: Limited flexibility for complex builds. **Ant**: Flexible but lacks dependency management by default.	Gradle combines flexibility with powerful **dependency management** (works with Maven Central, Ivy, Google repos, etc.).
Maven: Slower builds due to rebuilding everything. **Ant**: No incremental build support.	Gradle uses **incremental builds**, **build caching**, and **parallel execution** to make builds faster.
Maven: Works best for Java but not great for other languages. **Ant**: Can work with other languages but requires a lot of manual setup.	Gradle supports **Java, Kotlin, Groovy, Scala, C/C++, Android, and more** out of the box.
Maven & Ant: Harder to manage large multi-module projects.	Gradle has **first-class multi-project build support**, making it easier to scale for big projects.
Maven & Ant: Limited or outdated plugin ecosystems.	Gradle has a **rich plugin system** and allows custom plugins in Groovy, Kotlin, or Java.

Features of Gradle:

Flexibility:

Gradle uses a DSL (Domain Specific Language) based on Groovy or Kotlin, which allows users to define custom build logic. This is more flexible compared to XML-based configurations like Maven's pom.xml.

Incremental Builds:

Gradle supports incremental builds, meaning it can detect what parts of the build process need to be re-executed. This results in faster build times, as only the changed parts of the project are rebuilt, reducing the overall build time.

Dependency Management:

Gradle provides robust support for managing dependencies and repositories (like Maven Central, JCenter, or custom repositories). Gradle will automatically download the dependencies, manage their versions, and ensure compatibility.

Multi-project Builds:

Gradle is efficient in multi-project builds, where large projects can be split into several smaller sub-projects. It allows to define dependencies between these sub-projects and perform tasks on them in parallel to speed up builds.

Parallel Execution:

Gradle supports parallel execution of independent tasks. This is especially useful in large projects where tasks can be executed concurrently, dramatically speeding up build times.

Android Build System:

Gradle is the official build system for Android. Android Studio uses Gradle to handle the building, testing, and packaging of Android apps. It allows you to configure different build variants (e.g., development, production, etc.) easily.

Build Caching:

Gradle includes a build cache that can store task outputs and reuse them in future builds, even across different machines. This further speeds up builds, especially in CI/CD pipelines.

Extensibility:

Gradle can be easily extended through plugins. Plugins can be created and used to add

specific behaviour. Gradle has a rich ecosystem of plugins for popular tools and platforms like Docker, Java, Kubernetes, etc.

INSTALLATION OF GRADLE:

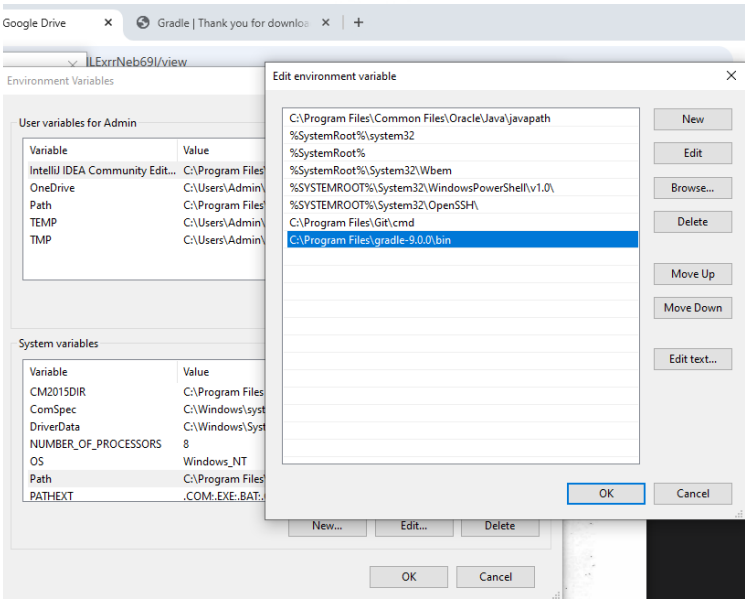
Gradle is a Java based tool. Hence the java 8 or higher version must be installed before installing Gradle.

Step 1: Visit the official page for Gradle https://gradle.org/install/

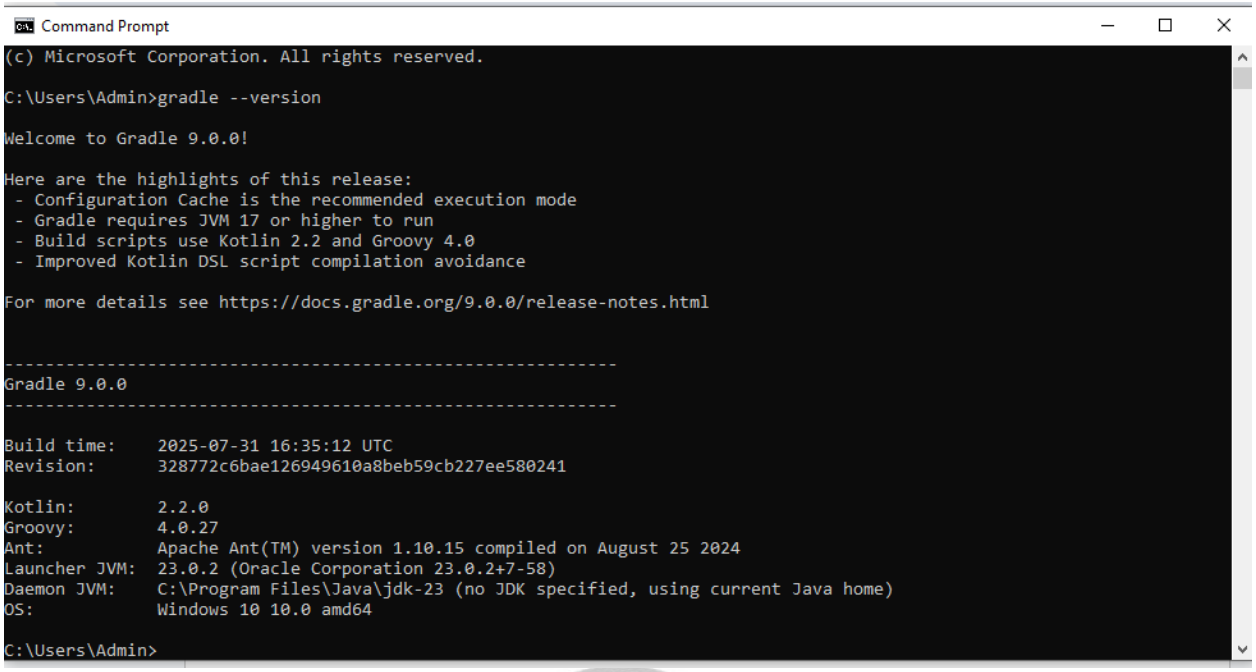
Step 2: Select the binary-only option for downloading the gradle zip file.

Step 3: Extract the file at the desired location.

Step 4: Configure the environment variable. Click on the path and Edit it by adding the path of gradle



Step 5: Open command prompt and issue the command gradle --version. It will display the version of gradle



Core Concepts:

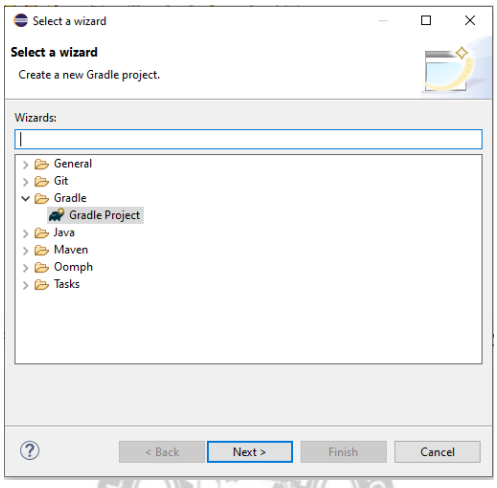
- **Project:** Gradle project represents the application that can be deployed.
- **Task:** A task refers to the piece of work performed by a build. For Example: the task can be creating jar files, Compiling classes, making javaDoc, etc.
- **Build Script:** Gradle builds the build script for project and task. Every gradle build represents one or more projects. The build script is written using the domain-specific language called Groovy. This script is saved as **build.gradle**

BUILDING BASIC APPLICATION USING GRADLE

Gradle application projects can be created by using IDE or using the command prompts.

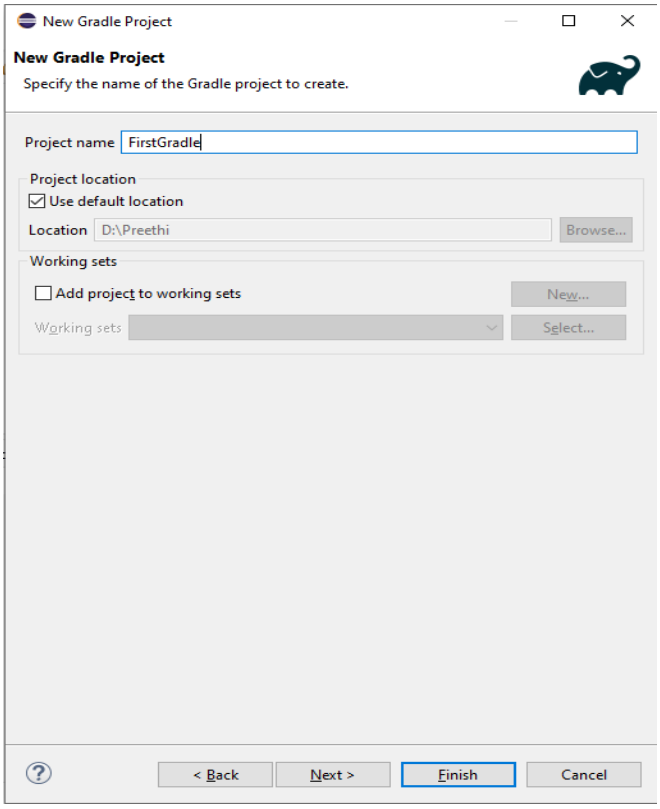
Using IDE

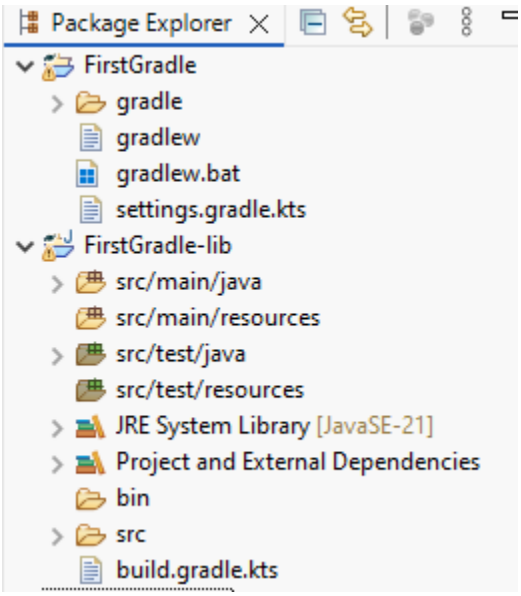
To create a new Gradle project with the Eclipse IDE, select the File New Other… menu entry and select Gradle/Gradle Project.



Click on the Next > button.

Enter the name of the Gradle project and Click Finish

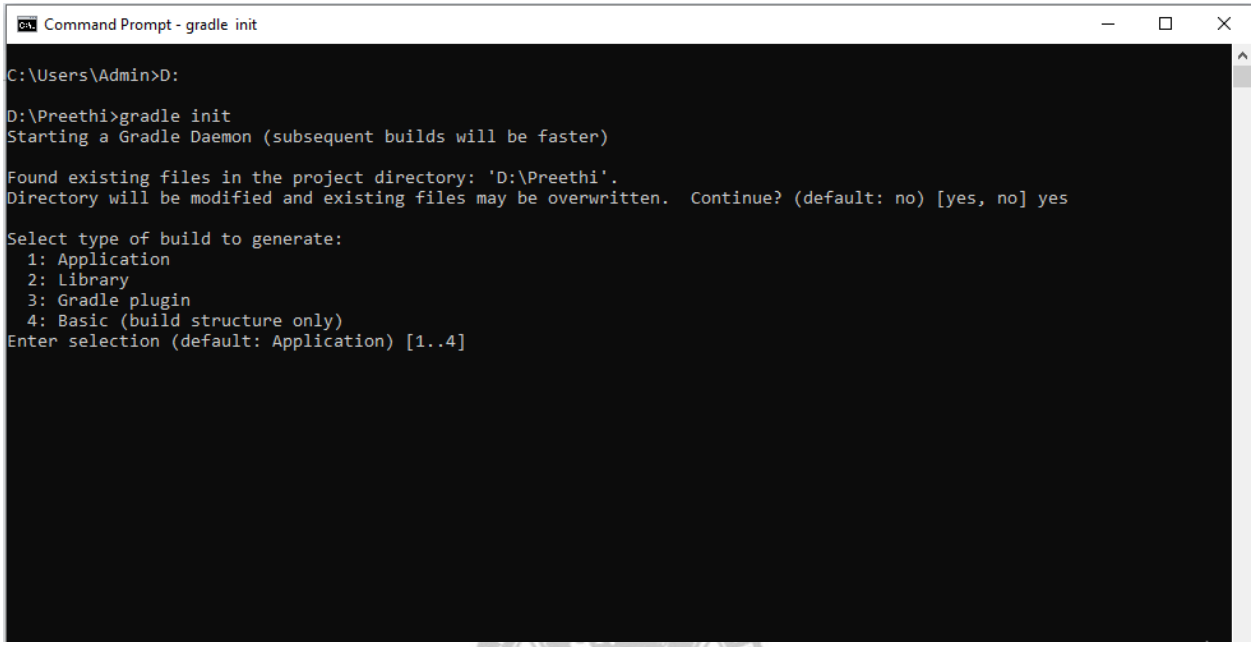




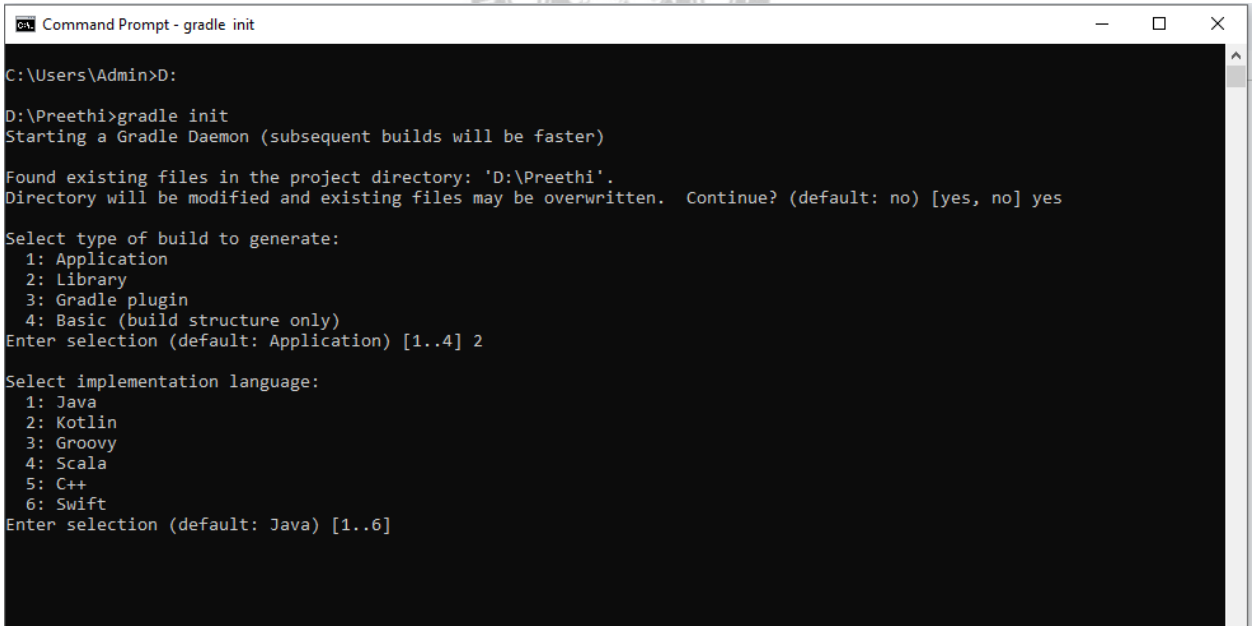
A new Gradle project will be created with the above mentioned directories.

Creating Gradle using Command prompt:

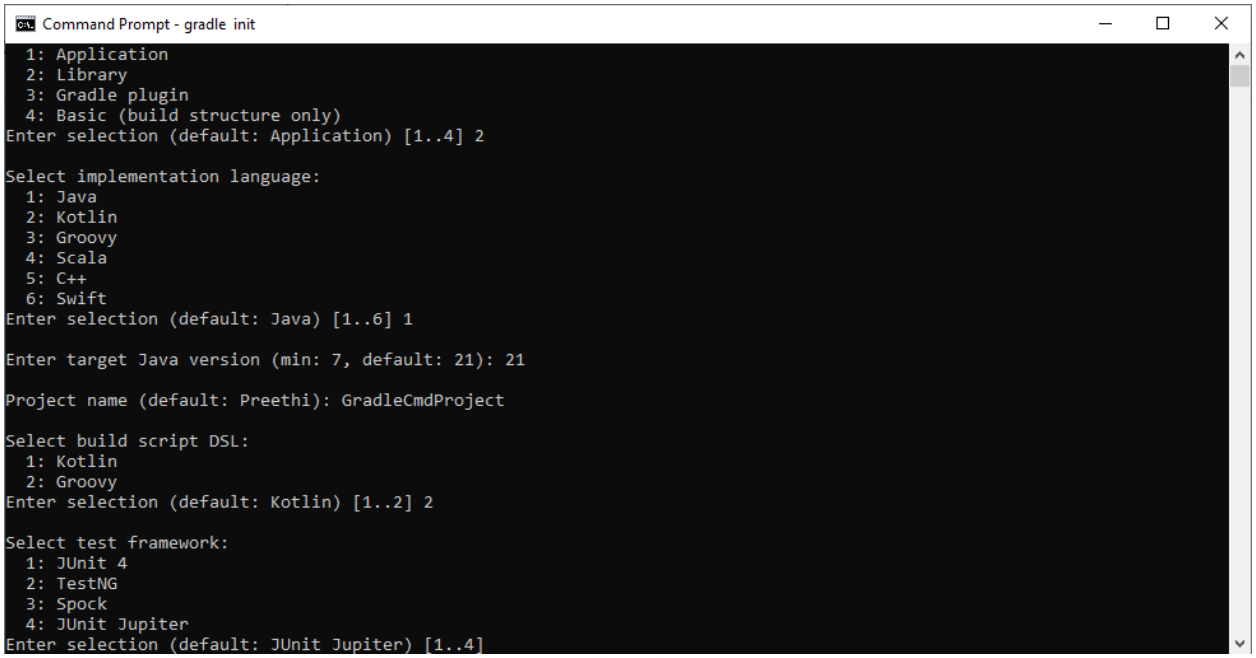
We can create Gradle project using command prompt using the command-line by missing the command **gradle init**



Enter the build type as 2 for application



Enter 1 for selecting Java and Enter the default version and then the name of the project. Select the Build Script as Groovy and select the test Framework as 1



Enter no to generate build using new APIs

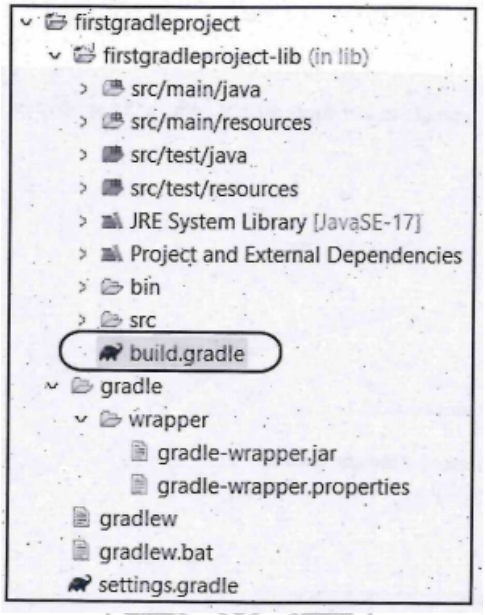
Thus the build directory gets created by above commands and app.jar gets generated.

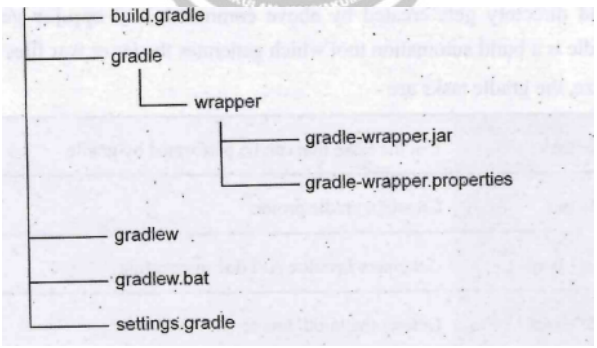
Gradle tasks:

1. **gradle tasks** : - List the tasks that can be performed by gradle
2. **gradle init** :- Creates a gradle project
3. **gradle javadoc** :- Generates JavaDoc API documentation

4. **gradle clean** :- Deletes the build directory
5. **gradle build** :- Compiles, unit test and packages the application
6. **gradle run** :- It executes the main class

Understanding Build in gradle:





Build.gradle:

Scripting file for build is written in build.gradle. It is similar to pom.xml file in Maven. The build.gradle file contains all the dependencies required to execute the project. It contains all configurations of the project. Gradle uses Domain Specific language (Groovy) for describing builds. All the tasks and plugins are defined in this file.

When the run command is called, it executes the build.gradle file in the current directory. The build file consists of three default sections: plugins, repositories and dependencies.

- **plugins:** We can apply the java-library plugin to add support for java library
- **Repositories:** We can declare internal and external repositories for resolving dependencies. We can declare different types of repositories supported by Gradle, Maven, Ant and Ivy.
- **Dependencies:** We can declare dependencies that are necessary for the project.

build.gradle

```
plugins {
  // Apply the java-library plugin for API and implementation separation.
  `java-library`
}
repositories {
  // Use Maven Central for resolving dependencies.
  mavenCentral()
}
dependencies {
  // Use JUnit Jupiter for testing.
  testImplementation(libs.junit.jupiter)
}
tasks.named<Test>("test") {
  // Use JUnit Platform for unit tests.
  useJUnitPlatform()
}
application{
//Define the main class for the application
}
```

gradle directory:

This is a directory located at the **root** of a Gradle project. It has a sub directory wrapper. Wrapper consists the following important files

- gradle-wrapper.jar - contains code for downloading the Gradle distribution specified in the gradle-wrapper.properties file
- gradle-wrapper.properties - contains wrapper runtime properties.

gradlew:

It is the script that executes Gradle tasks with the Wrapper on Linux and MacOS machines.

gradlew.bat:

It is the gradlew equivalent batch script for Windows machines

settings.gradle:

The settings.gradle file is used to configure the global settings of the gradle project.

The main benefit of the wrapper is that if a gradle project is being build on one machine we can use the gradle projects in another machine without installing gradle software.
