# Experiment 3: Working with Gradle: Setting Up a Gradle Project, Understanding Build Scripts (Groovy and Kotlin DSL), Dependency Management and Task Automation

# Setting Up a Gradle Project

## Step-by-Step Instructions

**Step 1: Ensure Gradle is Installed**

Before starting, verify that Gradle is installed on your system. Open a terminal and run:

gradle -v

.

Step 2: Create a New Gradle Project

To create a basic Java application project:

Step 1: Open IntelliJ IDEA and Create a New Project

1. Click on "New Project".

2. Select "Gradle" (under Java/Kotlin).

3. Choose Groovy or Kotlin DSL (Domain Specific Language) for the build script.

4. Set the Group ID (e.g., com.example ).

5. Click Finish.

Step 3: Explore the Project Structure **(take screen shot and paste this project structure on left side of record)**

After project creation, the directory structure typically looks like this:

HelloGradle/

├── build.gradle // The primary build script (Groovy DSL by

default)

├── gradle/ // Contains Gradle wrapper files (if generated)

├── gradlew // Unix shell script to run Gradle wrapper

├── gradlew.bat // Windows batch script for Gradle wrapper

├── settings.gradle // Contains project settings and names

└── src

├── main

│ └── java

│ └── App.java // Your main application source file

└── test

└── java

└── AppTest.java // Your test cases

## Explanation of Components:

### build.gradle:

This is the main build script written in Groovy (or Kotlin if you choose). It defines plugins, repositories, dependencies, and tasks.

### settings.gradle:

A small script that defines the project’s name and, in multi-project builds, the included subprojects.

* **gradlew/gradlew.bat:** The Gradle wrapper scripts. They allow you to run Gradle without requiring a separate installation on every machine by automatically downloading the correct Gradle version.

### src/main/java:

Contains your application’s source code.

### src/test/java:

Contains your unit tests.

# Understanding Gradle Build Scripts

A **build script** in Gradle is a programmatic file that instructs Gradle on how to build your project. It can be written in two main DSLs:

* + **Groovy DSL (build.gradle):** The traditional and most common syntax.
  + **Kotlin DSL (build.gradle.kts):** A statically-typed alternative that leverages Kotlin’s language features.

## Groovy DSL Example (build.gradle)

Below is an example of a basic build.gradle for a Java application using Groovy DSL

plugins {

// Apply the Java plugin for compiling Java code id 'java'

// Apply the application plugin to add support for building an application

id 'application'

}

group = 'com.example' version = '1.0'

repositories {

// Use Maven Central for resolving dependencies. mavenCentral()

}

dependencies {

// Define your dependencies. For example, JUnit for testing: testImplementation 'junit:junit:4.13.2'

}

application {

// Define the main class for the application. mainClass = 'com.example.App'

}

// A custom task example: printing a greeting task hello {

doLast {

println 'Hello, Gradle!'

}

}

### Explanation:

* **plugins block:** Declares the plugins used. The java plugin adds Java compilation and testing tasks, while the application plugin adds tasks for running the application.
* **group/version:** Sets project coordinates.
* **repositories block:** Configures the repository (Maven Central) where dependencies will be resolved.
* **dependencies block:** Lists the libraries your project depends on.
* **application block:** Specifies the main class of your application.
* **Custom task:** Defines a task named hello that prints a greeting when run.

## Kotlin DSL Example (build.gradle.kts)

Here’s how the same build configuration might look in Kotlin DSL:

Create a file named build.gradle.kts (or convert your file) with the following content:

plugins {

// Apply the Java plugin for compiling Java code java

// Apply the application plugin to add support for building an application

application

}

group = “com.example" version = "1.0"

repositories { mavenCentral()

}

dependencies {

// Define dependencies using Kotlin DSL syntax testImplementation("junit:junit:4.13.2")

}

application {

// Set the main class for the application mainClass.set("com.example.App")

}

// A custom task example using Kotlin DSL tasks.register("hello") {

doLast {

println("Hello, Gradle with Kotlin DSL!")

}

}

### Explanation:

* **Kotlin DSL Syntax:** Uses a statically typed syntax, which can be more intuitive for developers familiar with Kotlin.
* **Plugins, Repositories, and Dependencies:** Defined similarly to the Groovy DSL but with Kotlin-style function calls and property access.
* **Custom Task Registration:** Uses tasks.register to define a task, similar in functionality to the Groovy DSL.

# Task Automation in Gradle

Gradle comes with many built-in tasks (provided by plugins) and allows you to define your own custom tasks.

## Built-in Tasks:

* + **compileJava:** Compiles the source code in src/main/java.
  + **test:** Runs tests in src/test/java.
  + **jar:** Packages compiled code into a JAR file.
  + **run:** Runs the application (if the application plugin is applied).

}

*Screenshot Tip:* Capture your terminal output after running these custom tasks.

# Running and Verifying Your Gradle Project

## Common Gradle Commands:

**Build the Project:**

**Compile, Test, and Package:**

* + **Command:**

gradle build

### What it does:

* + - Compiles source code (compileJava).
    - Runs tests (test).
    - Packages the application into a JAR file (jar).

### Expected Output:

Look for a "BUILD SUCCESSFUL" message.

* + **Screenshot Tip:** Capture the full output of the gradle build command

### Running the Application Run **Command:**

gradle run: Runs your main class as specified in the application block.

### Expected Output: Any output from your application