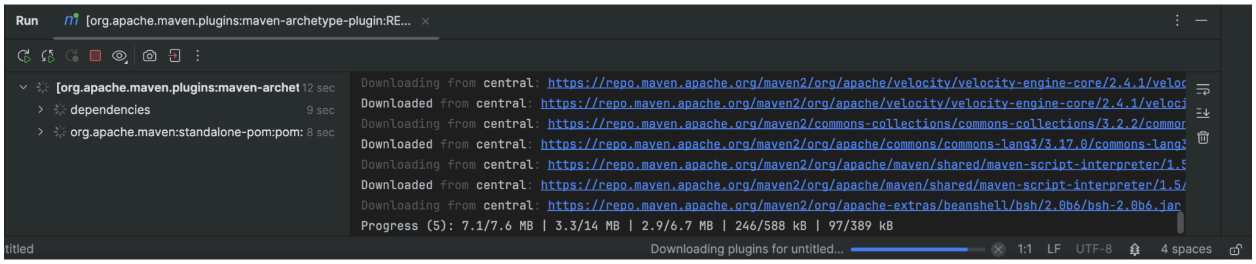
4. Practical Exercise: Build and Run a Java Application with Maven, Migrate the Same Application to Gradle

This exercise will guide you through building a basic Java application using Maven and then demonstrate the steps to migrate the same application to a Gradle build.

Part 1: Building and Running a Java Application with Maven

Step 1: Set Up the Maven Project Structure

1. **Create a Project Directory:** Create a new directory on your system named `maven-java-app`.



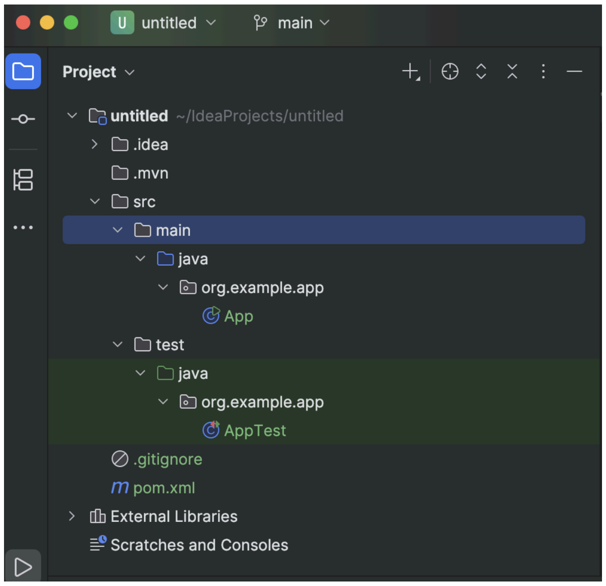
```
[org.apache.maven.plugins:maven-archet 12 sec
  dependencies 9 sec
  org.apache.maven:standalone-pom:pom: 8 sec
```

```
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/velocity/velocity-engine-core/2.4.1/veloc
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/velocity/velocity-engine-core/2.4.1/veloci
Downloading from central: https://repo.maven.apache.org/maven2/commons-collections/commons-collections/3.2.2/commor
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-lang3/3.17.0/commons-lang3
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-script-interpreter/1.5
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-script-interpreter/1.5/
Downloading from central: https://repo.maven.apache.org/maven2/org/apache-extras/beanshell/bsh/2.0b6/bsh-2.0b6.jar
Progress (5): 7.1/7.6 MB | 3.3/14 MB | 2.9/6.7 MB | 246/588 kB | 97/389 kB
```

Run
[org.apache.maven.plugins:maven-archetype-plugin:RE...
titled
Downloading plugins for untitled...
1:1
LF
UTF-8
4 spaces

Create Source Directories: Inside `maven-java-app`, create the following directory structure:

```
maven-java-app/
└── src/
    └── main/
    └── java/
        └── com/example/
```



```
untitled ~/IdeaProjects/untitled
  .idea
  .mvn
  src
    main
      java
        org.example.app
          App
    test
      java
        org.example.app
          AppTest
  .gitignore
  pom.xml
External Libraries
Scratches and Consoles
```

2. **Create a Java Source File:** Inside the `com/example/` directory, create a file named `App.java` with the following content:

   Java
```java
package com.example;


public class App {
    public static void main(String[] args) {
        System.out.println("Hello from Maven!");
    }
}
```

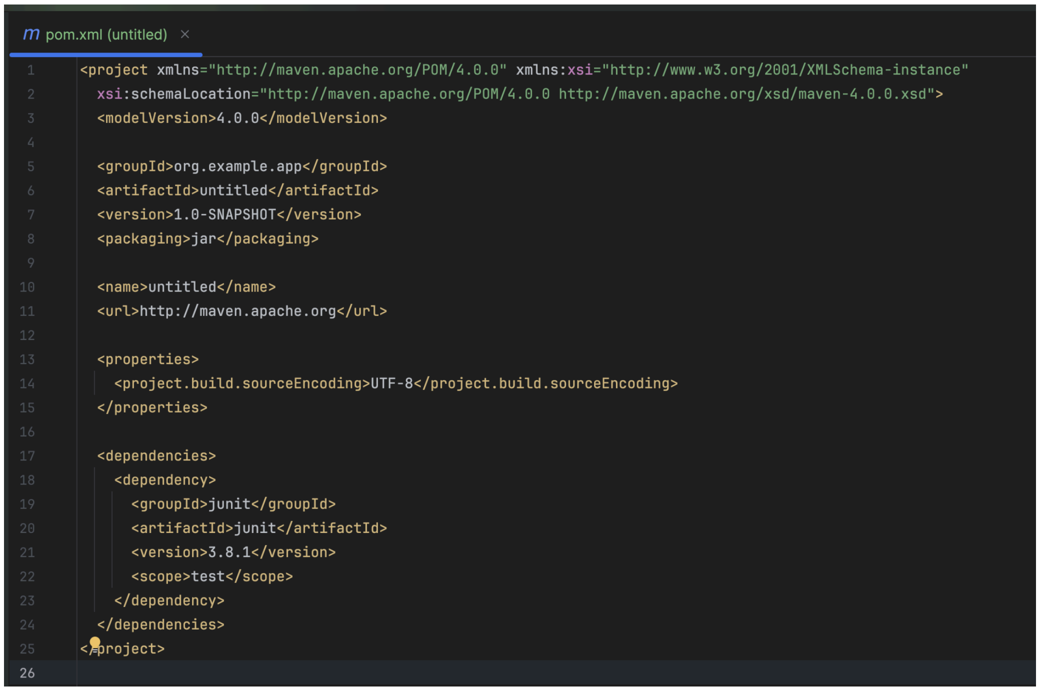
3. **Create a `pom.xml` File:** Inside the root `maven-java-app` directory, create a file named `pom.xml` (Project Object Model) with the following content:

   XML
```xml
<?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>maven-java-app</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>
</project>
```



```xml
<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>org.example.app</groupId>
  <artifactId>untitled</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>untitled</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

4. **Explanation of `pom.xml`:**
   - `<modelVersion>4.0.0</modelVersion>`: Specifies the Maven POM model version.
   - `<groupId>com.example</groupId>`, `<artifactId>maven-java-app</artifactId>`, `<version>1.0-SNAPSHOT</version>`: These define the unique coordinates of your project.
   - `<properties>`: Allows you to define project-wide properties. Here, we specify the Java source and target compatibility levels.

Step 2: Build the Maven Application

1. **Open Terminal or Command Prompt:** Navigate to the root `maven-java-app` directory in your terminal or command prompt.

Run the Maven Build Command: Execute the following Maven command:

Bash
```
mvn clean package
```

2. **Explanation of the Command:**
   - `mvn`: The Maven command-line tool.

    - `clean`: Deletes the `target` directory, which contains previous build outputs.
    - `package`: Compiles the source code and packages it into a JAR file.
3. Maven will download necessary dependencies (if any were declared), compile your `App.java` file, and create a JAR file (likely named `maven-java-app-1.0-SNAPSHOT.jar`) in the `target` directory.

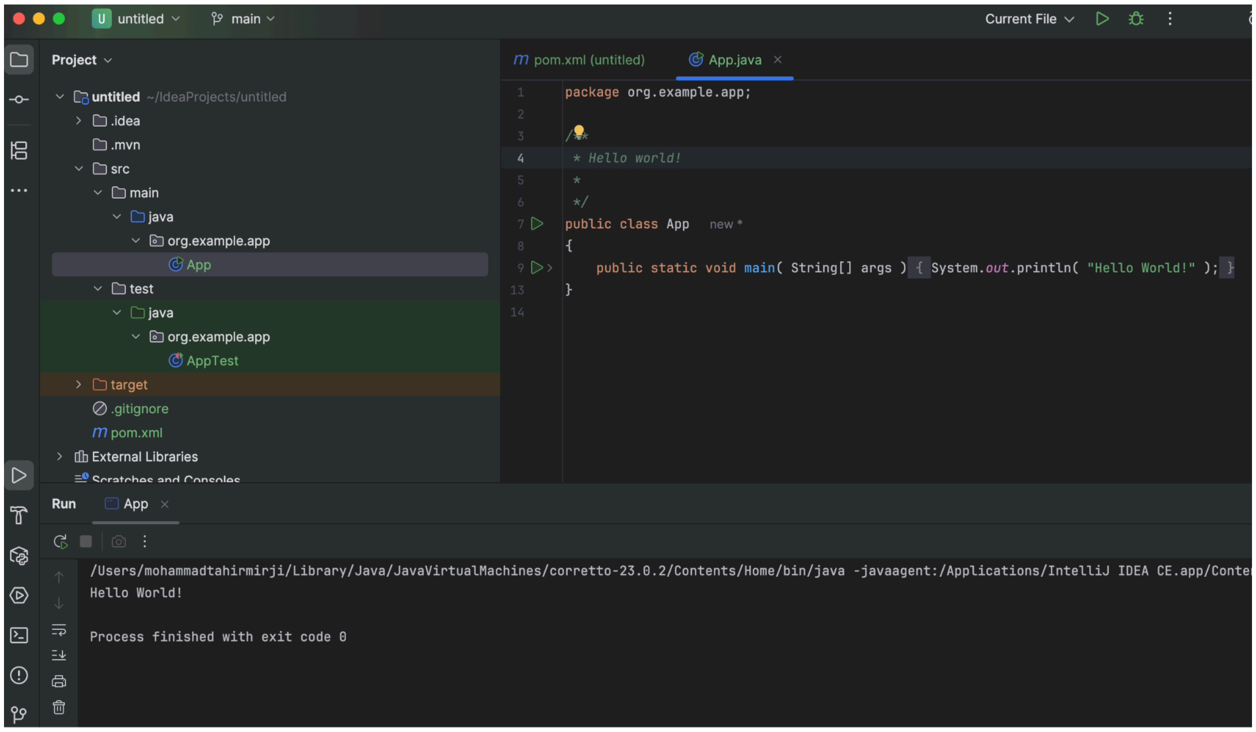
Step 3: Run the Maven Application

1. **Navigate to the `target` Directory:** Change your current directory in the terminal to the `target` directory inside `maven-java-app`.

Run the JAR File: Execute the following command to run your application:

Bash
```bash
java -jar maven-java-app-1.0-SNAPSHOT.jar
```
You should see the output:

```
Hello from Maven!
```



```java
package org.example.app;

/**
 * Hello world!
 *
 */
public class App   new *
{
    public static void main( String[] args ) { System.out.println( "Hello World!" ); }
}
```

```
/Users/mohammadtahirmirji/Library/Java/JavaVirtualMachines/corretto-23.0.2/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Conte
Hello World!

Process finished with exit code 0
```

Part 2: Migrating the Application to Gradle

Now, let's migrate the same simple application to a Gradle build.

Step 1: Set Up the Gradle Project Structure

1. **Create a New Directory:** Create a new directory named `gradle-java-app` (you can keep the `src` directory from the Maven project or create a new one inside this directory).

2. **Copy Source Files (Optional):** If you didn't create a new `src` directory, ensure the `src/main/java/com/example/App.java` file from the Maven project is present within `gradle-java-app`.

3. **Create Gradle Build Files:** Inside the root `gradle-java-app` directory, create the following files:

`build.gradle` **(for Groovy DSL):**

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

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

repositories {
    mavenCentral()
}

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

   ○

`build.gradle.kts` **(for Kotlin DSL):**

```
 Kotlin
plugins {
    java
    application
}

group = "com.example"
version = "1.0-SNAPSHOT"
```

```
repositories {
    mavenCentral()
}

application {
    mainClass.set("com.example.App")
}
```

○

`settings.gradle` (for Groovy DSL):

```
 Groovy
rootProject.name = 'gradle-java-app'
```

○

`settings.gradle.kts` (for Kotlin DSL):

```
 Kotlin
rootProject.name = "gradle-java-app"
```

4. **Explanation of Gradle Build Files:**
   - **`plugins { ... }`**:
     - `id 'java'` (Groovy) / `java` (Kotlin): Applies the Java plugin, providing Java compilation and testing capabilities.
     - `application`: Applies the Application plugin, which helps in creating runnable JVM applications and defines the `mainClass`.
   - **`group = '...'` / `group = "..."` and `version = '...'` / `version = "..."`**: Define the project's coordinates, similar to Maven.
   - **`repositories { ... }`**: Specifies where Gradle should look for dependencies (in this simple example, we don't have any external dependencies, but `mavenCentral()` is included as a standard practice).
   - **`application { mainClass = '...' / mainClass.set("...") }`**: Configures the Application plugin, specifying the fully qualified name of the main class to be executed.
   - **`settings.gradle` / `settings.gradle.kts`**: Defines the root project name.

Step 2: Build the Gradle Application

1. **Open Terminal or Command Prompt:** Navigate to the root `gradle-java-app` directory.

Run the Gradle Build Command: Execute the following Gradle command:

Bash
```
./gradlew   build
```

2. (On Windows, you might need to use `gradlew.bat build`)

   Gradle will download its wrapper dependencies (if it's the first time running in this project), compile your `App.java` file, and create a JAR file (likely in the `build/libs` directory).

Step 3: Run the Gradle Application

1. **Navigate to the Output Directory:** Change your current directory in the terminal to `build/libs` inside `gradle-java-app`.

Run the JAR File: Execute the following command to run the Gradle-built application:

Bash
```
java   -jar   gradle-java-app-1.0-SNAPSHOT.jar
```

2. **Run the JAR File:** Execute the following command to run the Gradle-built application:

   java -jar gradle-java-app-1.0-SNAPSHOT.jar

   You should see the same output:

   Hello from Gradle!