**2.**Working with Maven: Creating a Maven Project, Understanding the POM File, Dependency Management and Plugins.

**Solution:**

1: Install the Java JDK

If you haven’t installed the Java JDK yet, you can follow the link below to download and install it. Download Java JDK from Oracle

Working with Maven is a key skill for managing Java-based projects, particularly in the areas of build automation, dependency management, and project configuration. Below is a guide on creating a Maven project, understanding the POM file, and using dependency management and plugins:

**Overview of the Project**

2: Creating a Maven Project Using IDEs

Most modern IDEs (like IntelliJ IDEA or Eclipse) provide wizards to generate Maven projects. For example, in IntelliJ IDEA:

1. Go to **File > New Project**.
2. Choose **Maven** from the list of project types.
3. Provide the **groupId** and **artifactId** for your project.

**3: Understanding the POM File**

The **POM (Project Object Model)** file is the heart of a Maven project. It is an XML file that contains all the configuration details about the project. Below is an example of a simple POM file:

A basic **pom.xml** structure looks like this:



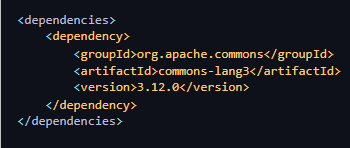
**Key element in pom.xml:**

* **<groupId>:**The group or organization that the project belongs to.
* **<artifactId>:**The name of the project or artifact.
* **<version>:**The version of the project (often follows a format like 1.0-SNAPSHOT).
* **<packaging>:**Type of artifact, e.g., jar, war, pom, etc.
* **<dependencies>:** A list of dependencies the project requires.
* **<build>:**Specifies the build settings, such as plugins to use.

**4: Dependency Management**

Maven uses the <dependencies> tag in the pom.xml to manage external libraries or dependencies that your project needs. When Maven builds the project, it will automatically download these dependencies from a repository (like Maven Central).

**Example of adding a dependency:**

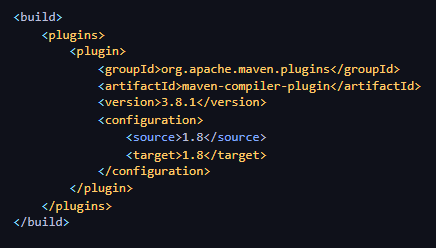


* **Transitive Dependencies**
  + Maven automatically resolves transitive dependencies. For example, if you add a library that depends on other libraries, Maven will also download those.
* **Scopes**
  + Dependencies can have different scopes that determine when they are available:
    - **compile** (default): Available in all build phases.
    - **provided**: Available during compilation but not at runtime (e.g., a web server container).
    - **runtime**: Needed only at runtime, not during compilation.
    - **test**: Required only for testing.

**5: Using Plugins**

Maven plugins are used to perform tasks during the build lifecycle, such as compiling code, running tests, packaging, and deploying. You can specify plugins within the <build> section of your **pom.xml**.

* **Adding Plugins**
  + You can add a plugin to your pom.xml like so

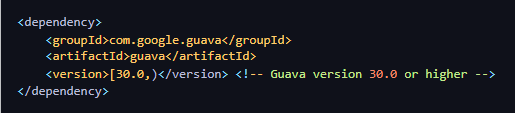


In this example, the **maven-compiler-plugin** is used to compile Java code and specify the source and target JDK versions.

1. **Common Plugins**
   * **maven-compiler-plugin**: Compiles Java code.
   * **maven-surefire-plugin**: Runs unit tests.
   * **maven-jar-plugin**: Packages the project as a JAR file.
   * **maven-clean-plugin**: Cleans up the target/ directory.
2. **Plugin Goals** Each plugin consists of goals, which are specific tasks to be executed. For example:
   * **mvn clean install:** This will clean the target directory and then install the package in the local repository.
   * **mvn compile:** This will compile the source code.
   * **mvn test:** This will run unit tests.

**6: Dependency Versions and Repositories**

1. **Version Ranges**
   * You can specify a version range for dependencies, allowing Maven to choose a compatible version automatically. for example:



1. **Repositories**
   * Maven primarily fetches dependencies from Maven Central, but you can also specify custom repositories. For example:



**Working with Maven Project**

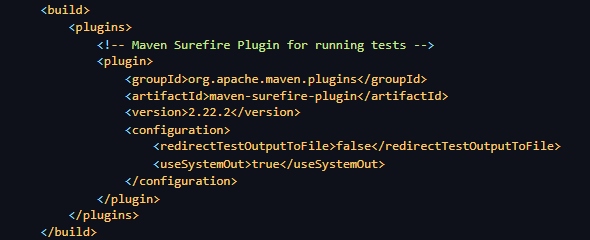
Note: Always create a separate folder to do any program.

Step 1: Creating a Maven Project

You can create a Maven project through your IDE.

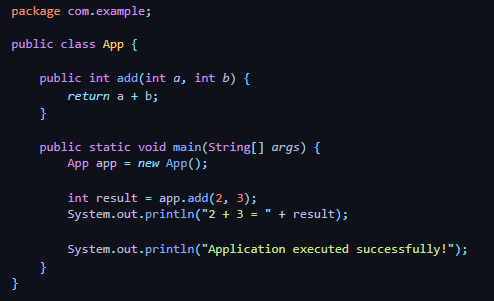
**Step 2: Open The pom.xml File**

* You can manually navigate the **project folder** named call **myapp** and open the file pom.xml and write the below code and paste it then save it.
* In case if you not getting project folder then type command in your cmd.
  + **cd myapp** – is use to navigate the project folder.
  + **notepad pom.xml** – is use to open pom file in notepad.



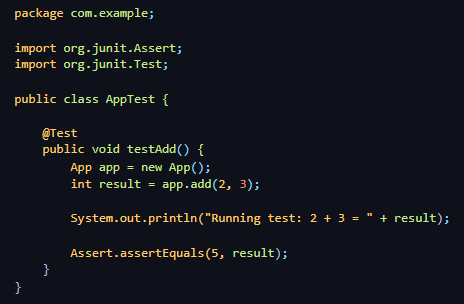
**Step 3: Open Java Code (App.java) File**

* Open a file **App.java** inside the **src/main/java/com/example/** directory.
* After opening the **App.java** write the below code and paste it in that file then save it.

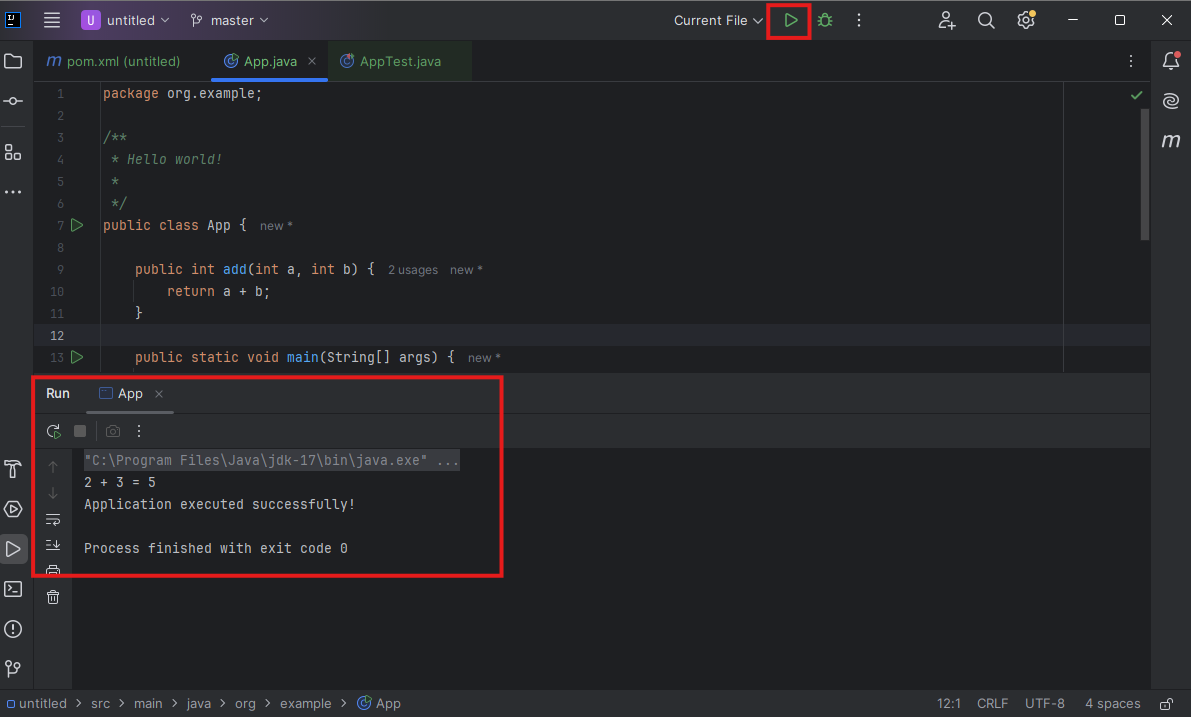


**Step 4: Open Java Code (AppTest.java) File**

* Open a file **AppTest.java** inside the **src/test/java/com/example/** directory.
* After opening the **AppTest.java** write the below code and paste it in that file then save it.



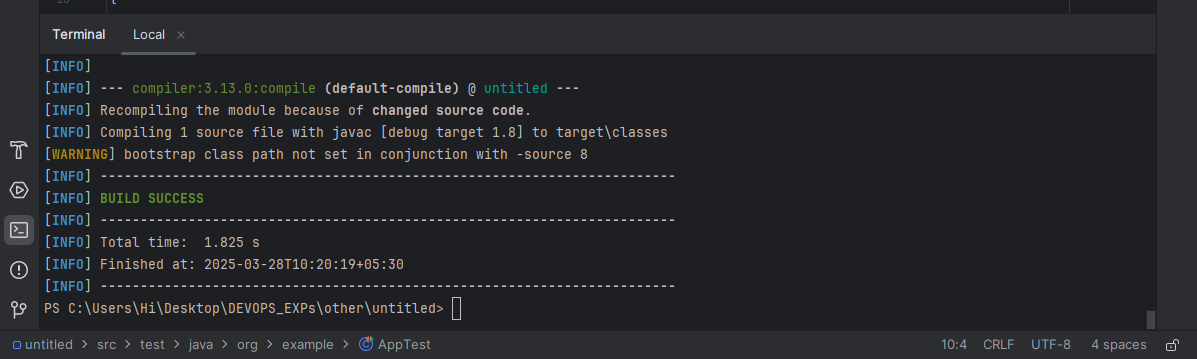
**Step 4: Building the Project**



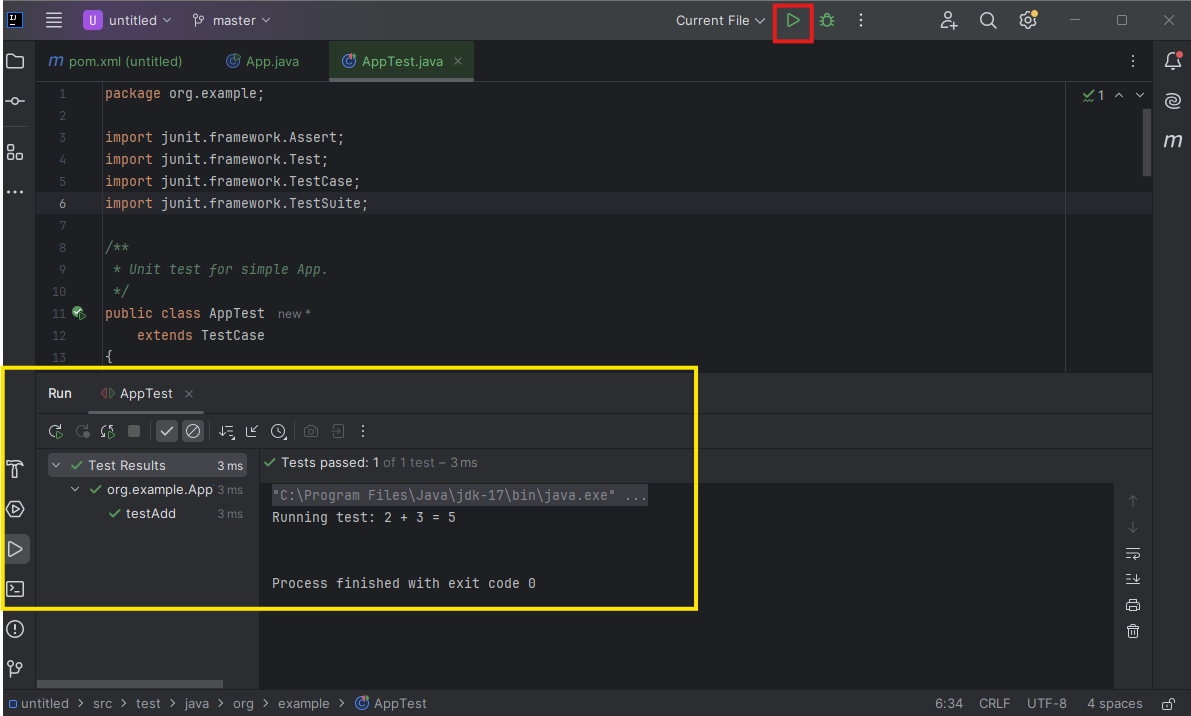
To build and run this project, follow these steps via commands or use green play button to execute:

1. **Compile the Project**



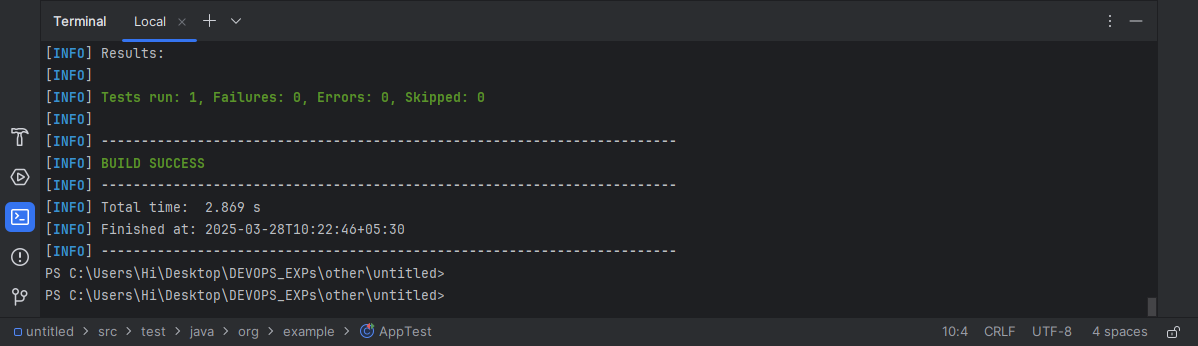


1. **Run the Unit Tests**



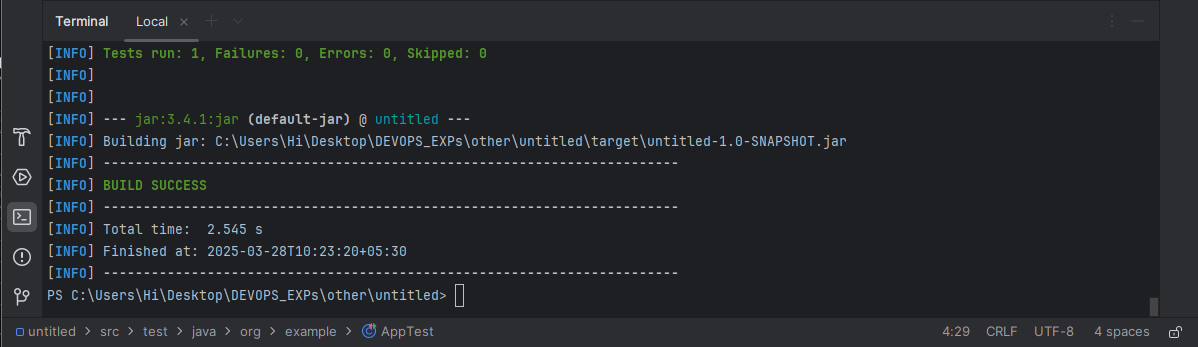
Or use the command below in the terminal to test





1. **Package the project into a JAR**

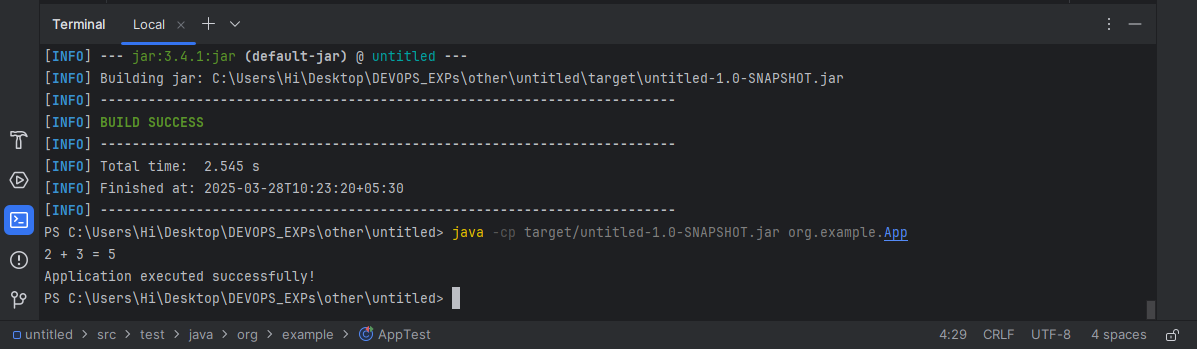


****

1. **Run the application (using JAR)**

Use the below command in the terminal to run jar.

**java -cp target/untitled-1.0-SNAPSHOT.jar org.example.App**

****