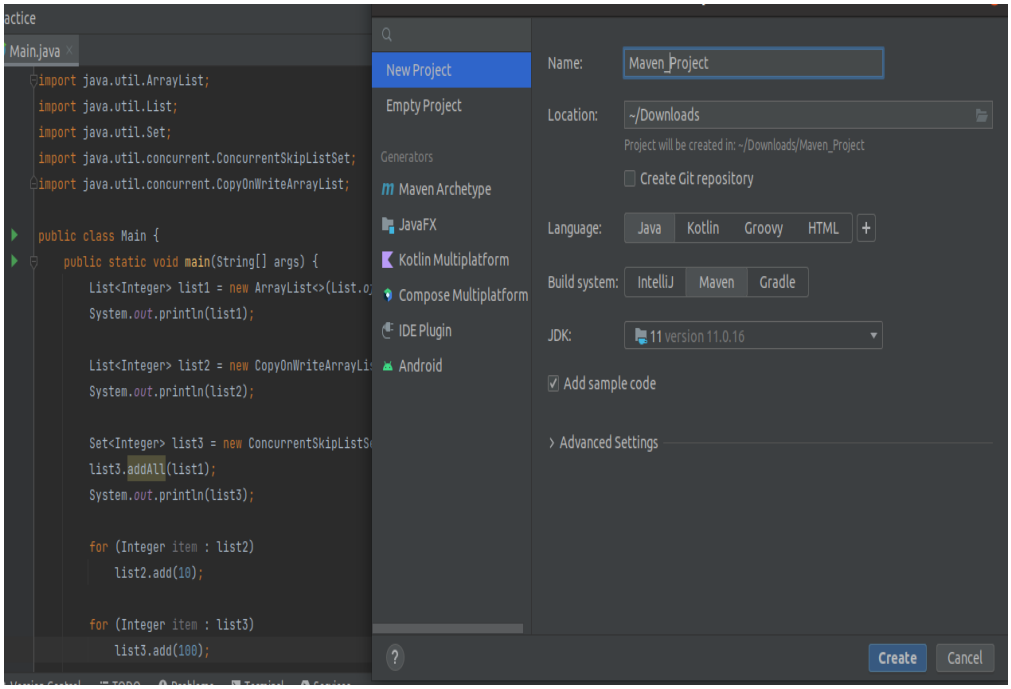
1. Add a maven dependency and its related repository URL.

   Before adding maven dependencies we have to create a maven project.

   Below are the steps for creating a maven project in java.

   Choose java as a language and the build system would be maven.

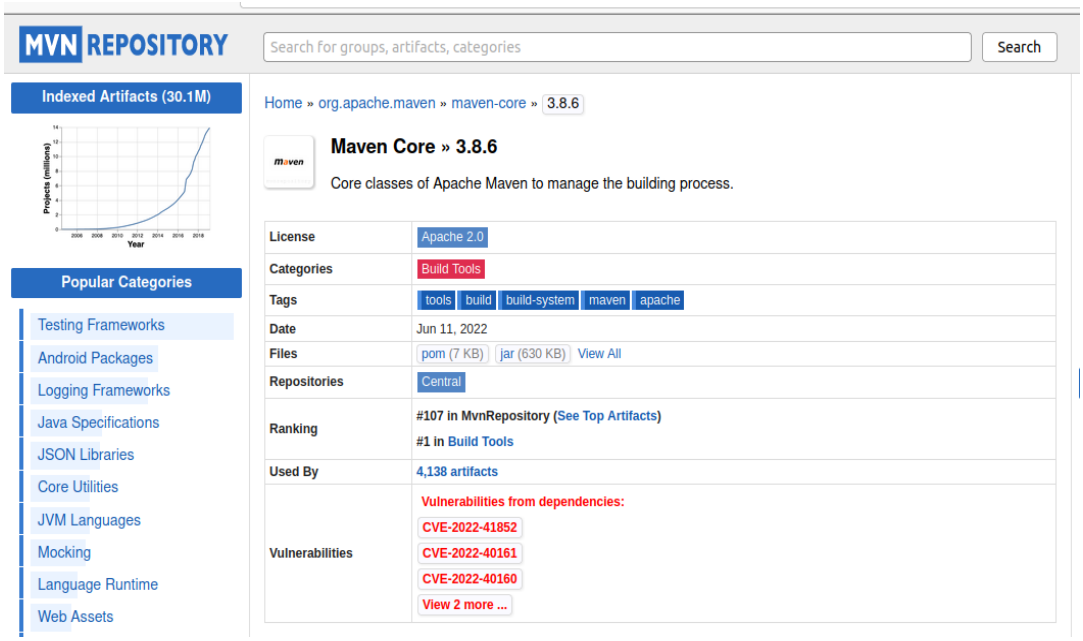


Dependency :- In Maven, a dependency is just **another archive—JAR, ZIP, and so on—which our current project needs in order to compile, build, test, and/or run**. These project dependencies are collectively specified in the pom. xml file, inside of a <dependencies> tag.

Maven Dependency injection **enables you to turn regular Java classes into managed objects and to inject them into any other managed object**. Using dependency injection, your code can declare dependencies on any managed object.

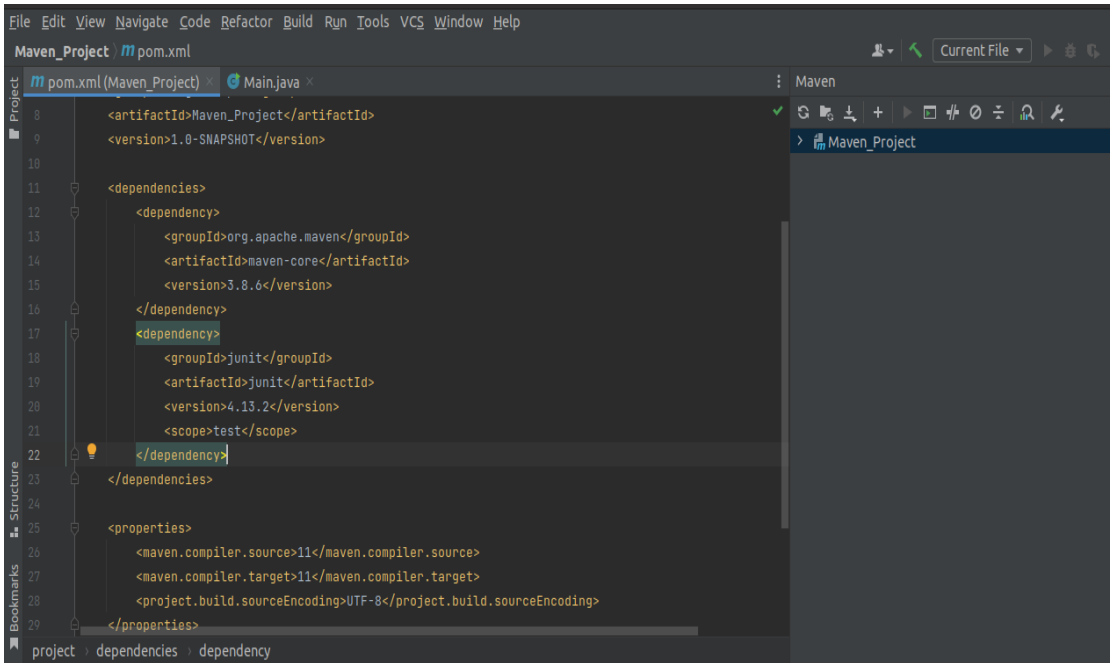
Now we want to add maven dependencies so we can add dependencies in the project.

Below are the site from which we have to download and inject the dependency in java.



Here is the download link for the maven-core dependency:

https://mvnrepository.com/artifact/org.apache.maven/maven-core/3.8.6



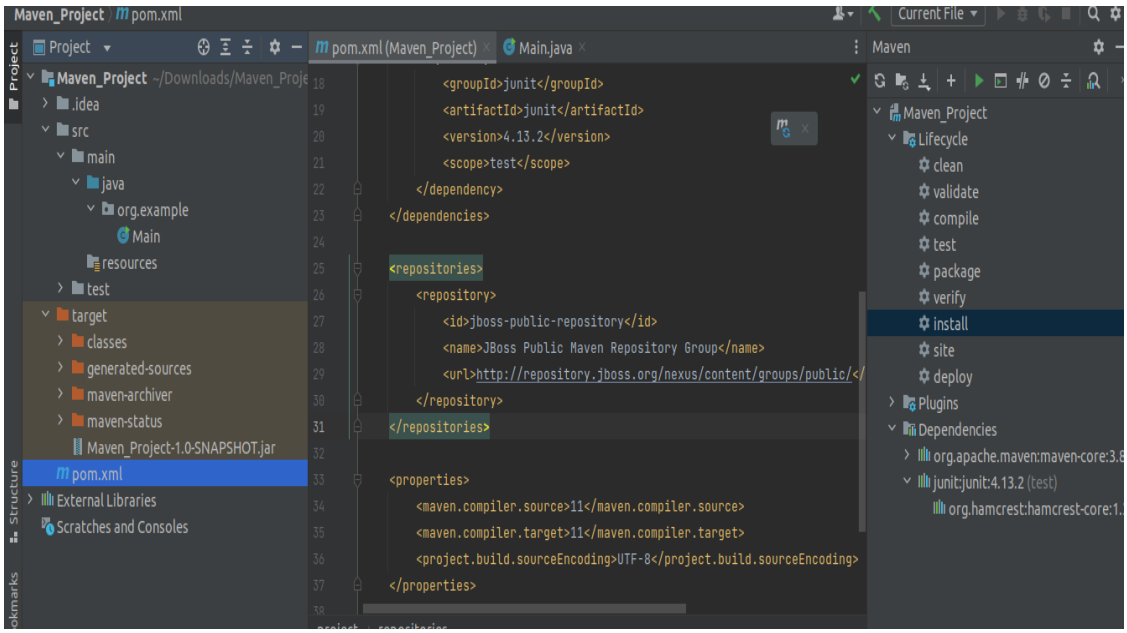
2. Add a new repository in the pom.xml and use its dependencies.

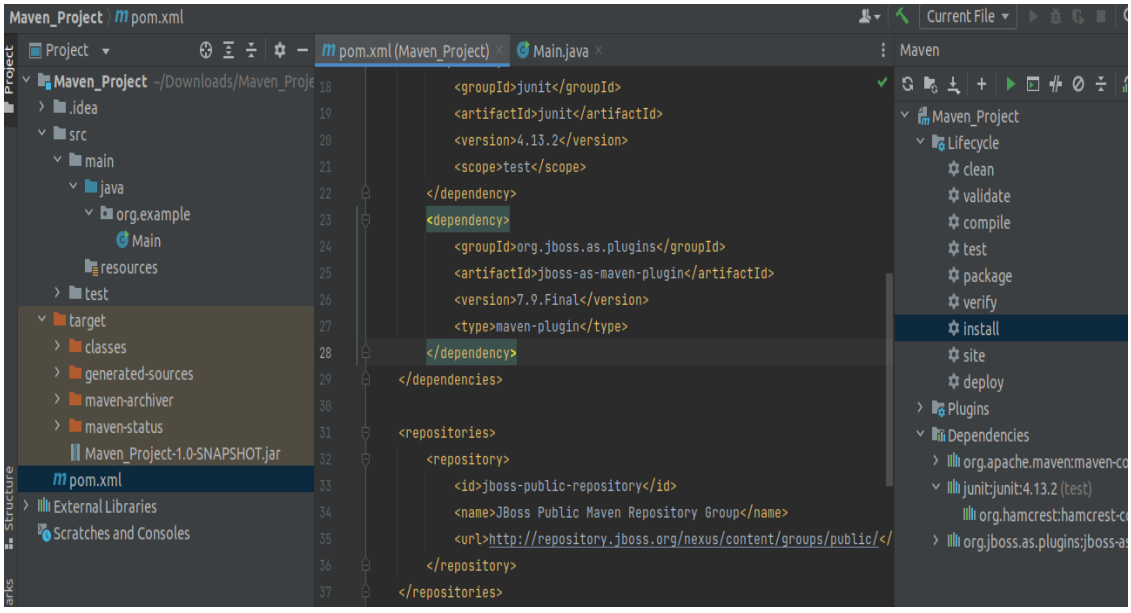
   Add New Repository :- We can add our own dependencies in the pom.xml file.

   There are two different ways that you can specify the use of multiple repositories.

   The first way is to specify in a POM which repositories you want to use.

   Below adding a j-boss repository and its dependency in the pom.xml file.

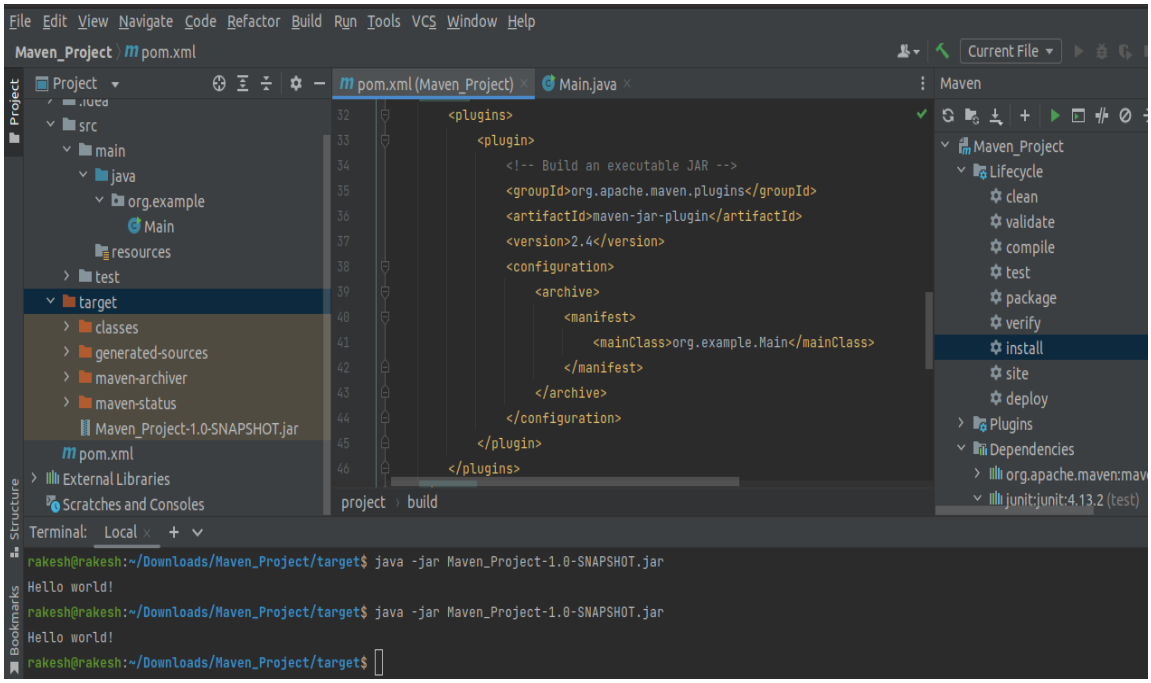




3. Using the JAR plugin, make changes in the pom.xml to make the jar executable. Using java -jar JAR_NAME, the output should be printed as "Hello World"

   Maven JAR plugin : we can add maven jar plugin in the pom.xml file to make a jar executable means we can run that jar in the terminal by using the below command

   java -jar JAR_NAME



   In the above example we printed “hello world” by running the executable file in the terminal.

   For archive this thing we have to put maven jar plugin in the pom.xml file and provide groupId, artifactId and the version.

   We have also provide the configuration of the class which will be executed by the jar

   So that we give the actual path of the class in the manifest tag which is under archive tag which is under configuration tag.

4. Differentiate between the different dependency scopes: compile, runtime, test, provided using different dependencies being defined in your pom.xml.

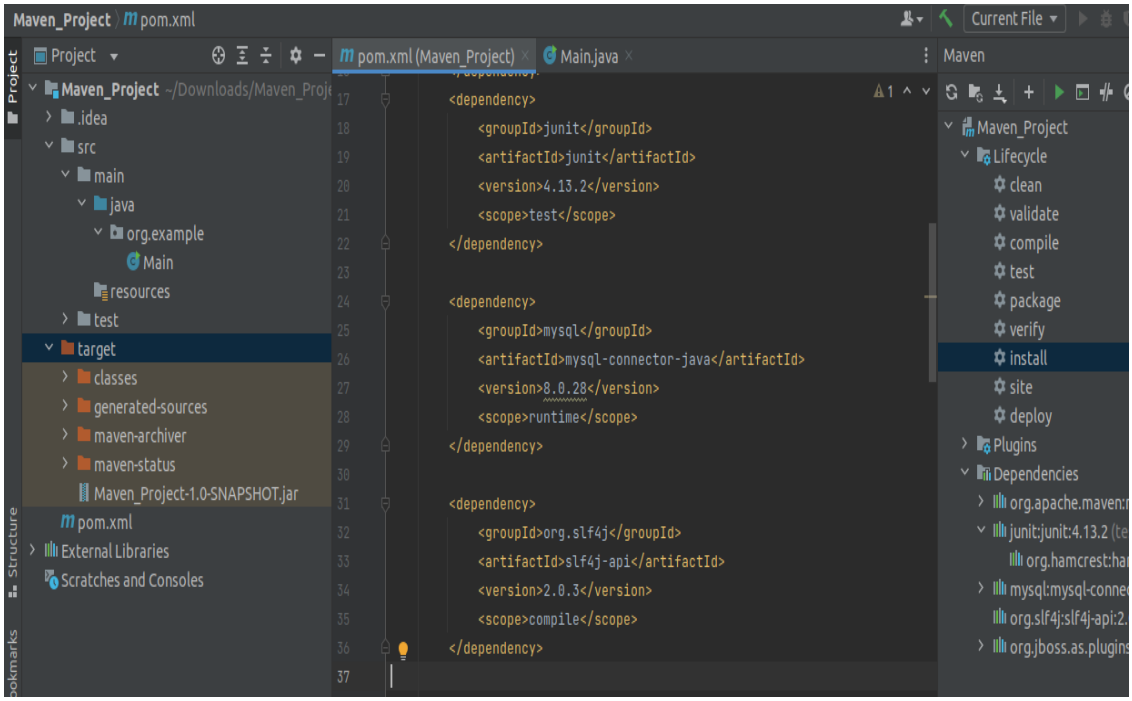
   Below are the differences between all the scope provided by maven.

   compile This is the default scope, used if none is specified. Compile dependencies are available in all classpaths of a project. Furthermore, those dependencies are propagated to dependent projects.

   runtime This scope indicates that the dependency is not required for compilation, but is for execution. It is in the runtime and test classpaths, but not the compile classpath.

   test This scope indicates that the dependency is not required for normal use of the application, and is only available for the test compilation and execution phases. This scope is not transitive.

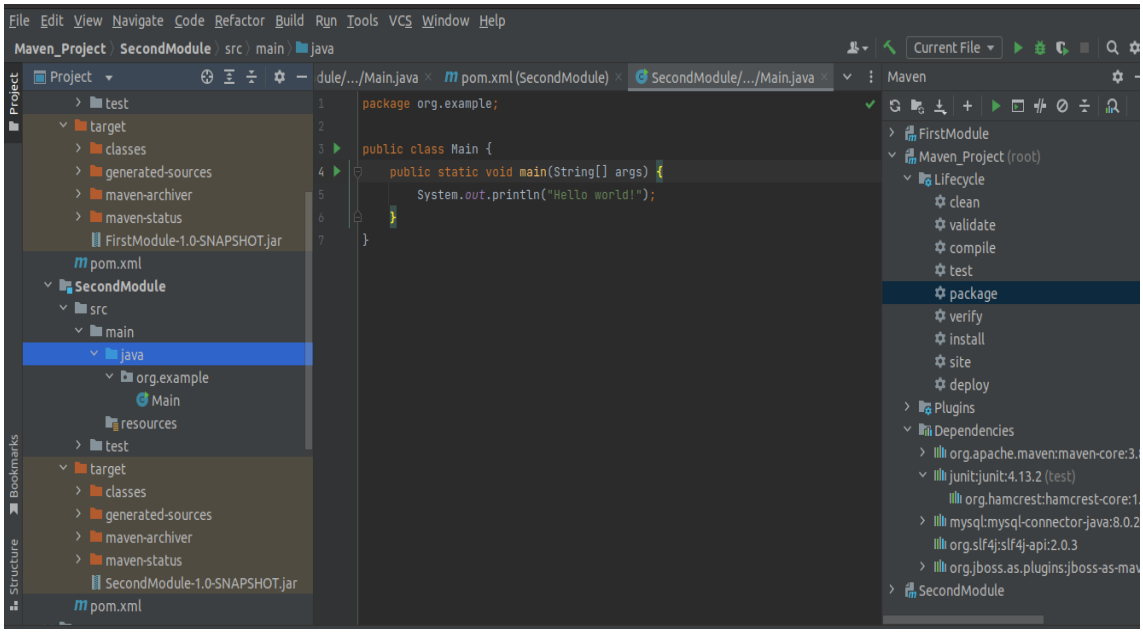
   provided This is much like compile, but indicates you expect the JDK or a container to provide the dependency at runtime. For example, when building a web application for the Java Enterprise Edition.



5. Create a multi-module project. Run the package command at the top level to make a jar of every module.

   In this question we have to create multiple module in the java project and run the

   package command at the top of the project for making the jar of each and every module.



   Now click the package option in the maven lifecycle after that all jars are created

   correspond to each and every module of the project.