Dependency version – 5:16 minute

Let’s learn how to specify the version of a dependency. We will also introduce you to version conflict resolution, which is how Gradle build tool resolves requests for multiple versions of the same dependency.

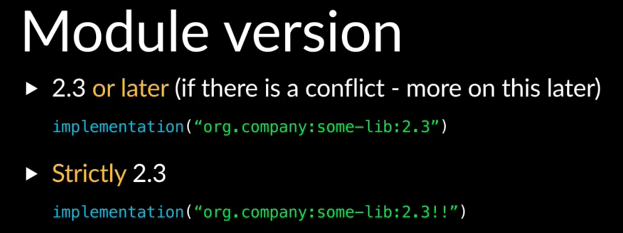
Dependency Management

Module version

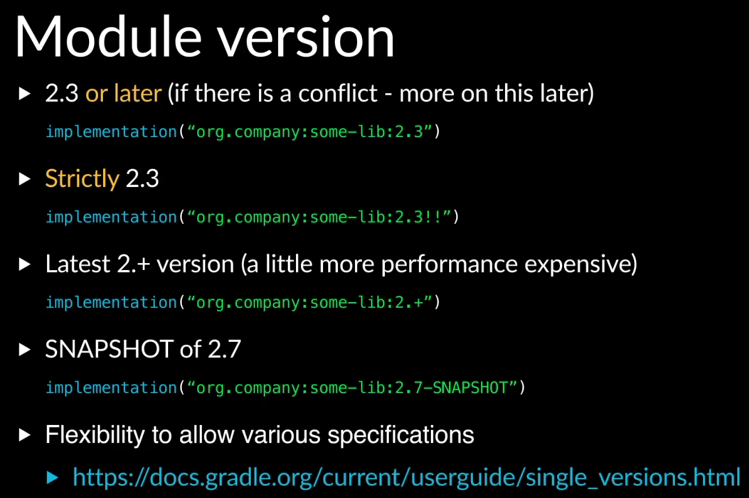
In addition to the module ID and bucket dependency configuration, you also need to specify what version of a dependency to use. Gradle build tool provides a lot of flexibility here. You can specify a single version or a range of versions, it’s important to understand that when you configure version 2.3 of library, that actually means at least version 2.3.



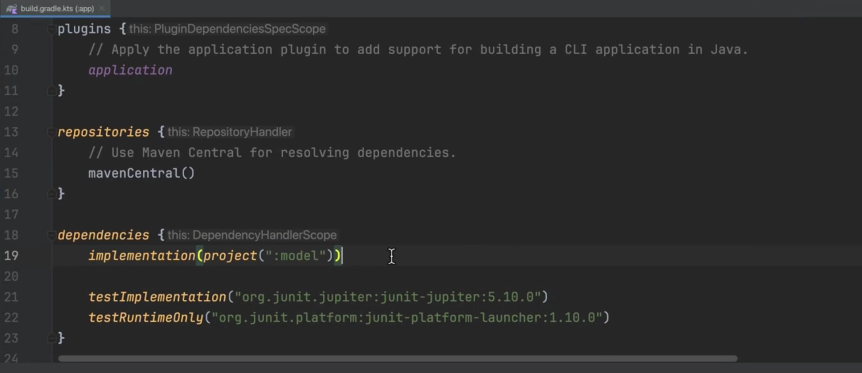
Gradle build tool may get a higher version, this could happen when another dependency also needed the library as a transitive dependency, but needed a higher version. We’ll take a look at an example of this in a bit. If you wanted 2.3 and 2.3 only, you can specify two exclamations after the version number then Gradle build tool will only get version 2.3.



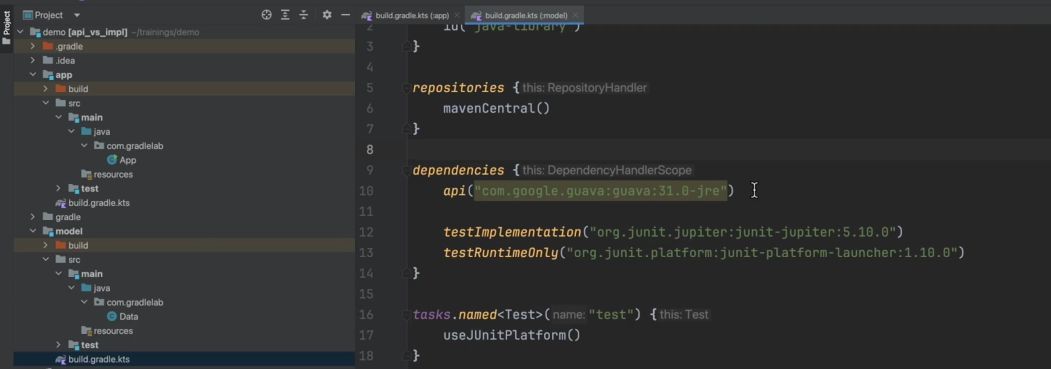
There are a lot of other options available when specifying dependency versions, which will be covered in another training. You can also refer to the docs to read up on these.



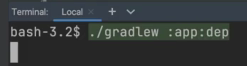
Let’s take a look at a simple example of what happens when the same dependency is requested twice with different versions. We look at our project with two subprojects again, app has a project dependency on model.

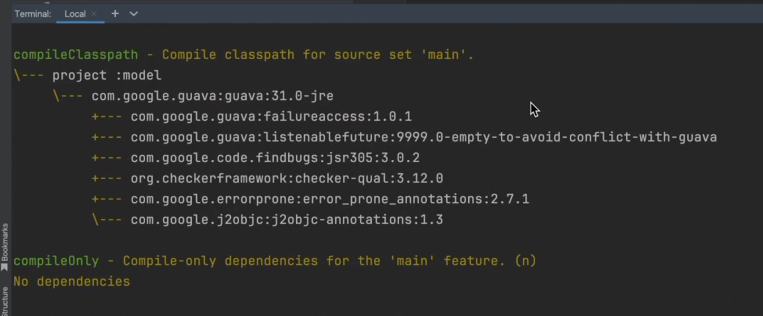


And model has an API dependency on guava.

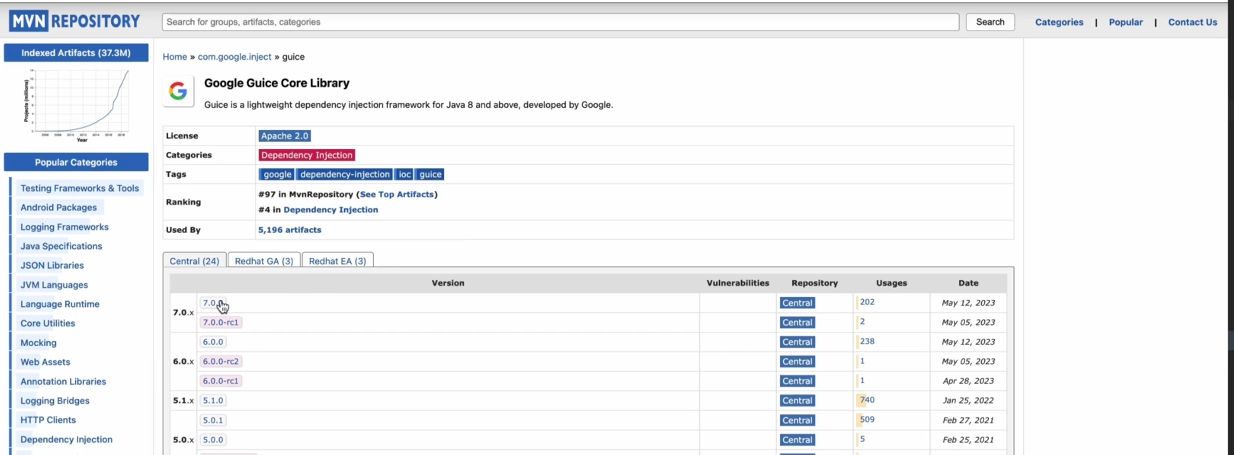


We’ve lowered the version of guava for this example to 31.0-jre. This means to use at least version 31.0-jre, if we look at apps dependencies, we can see guava shows up in the compiled class path

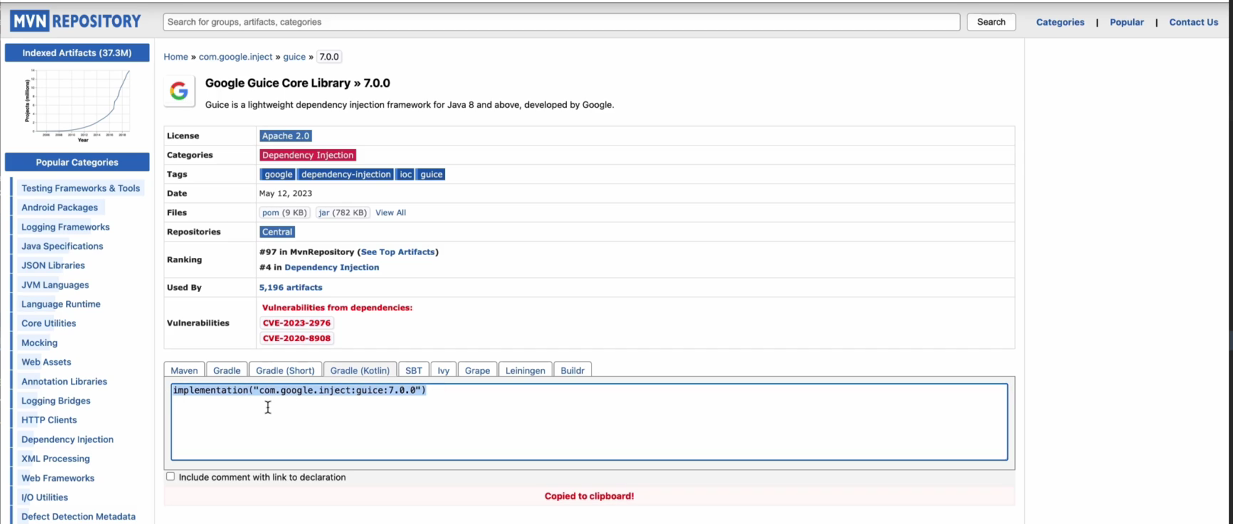




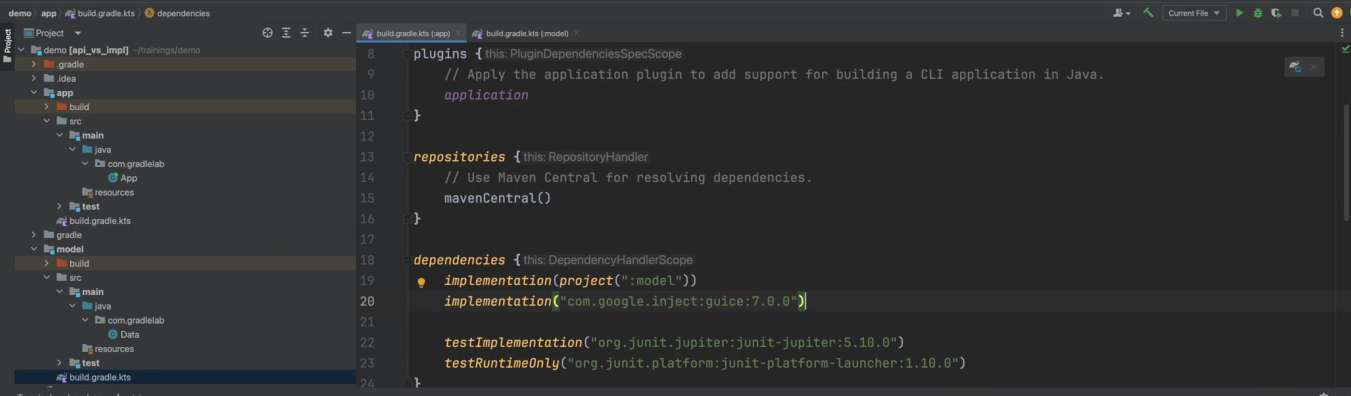
as a transitive dependency of model. Now let’s add a direct dependency to app with the guice version, we’ll pick the most recent version.



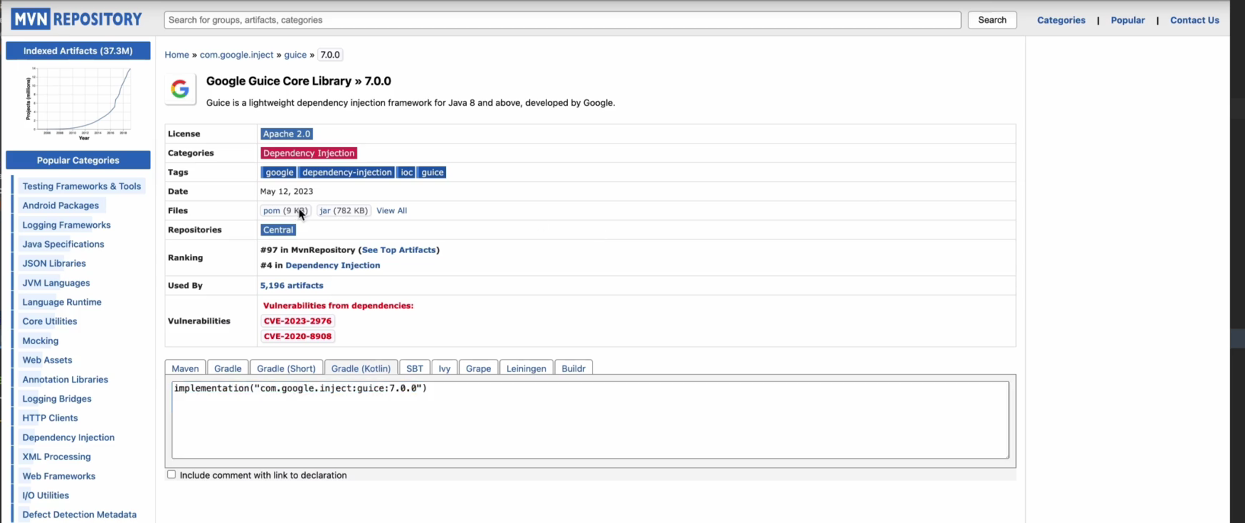
You can see mvnrepository.com, gives me a string directing copy paste.

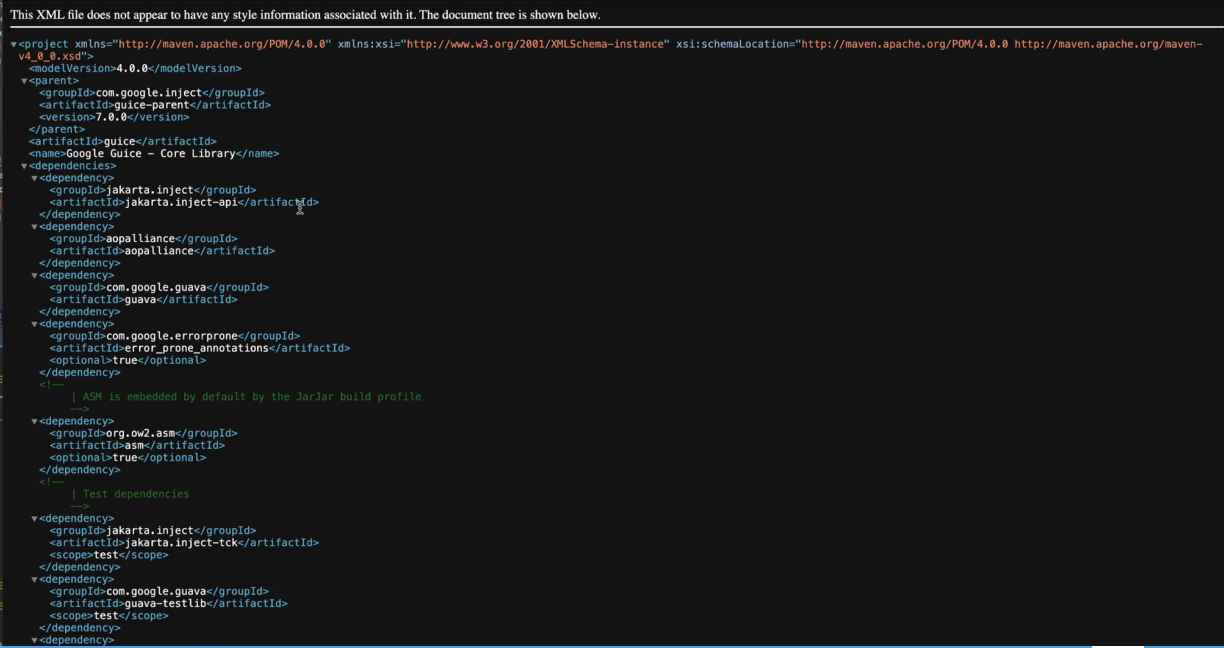


And using the configuration. Now, guice has a dependency on guava too,

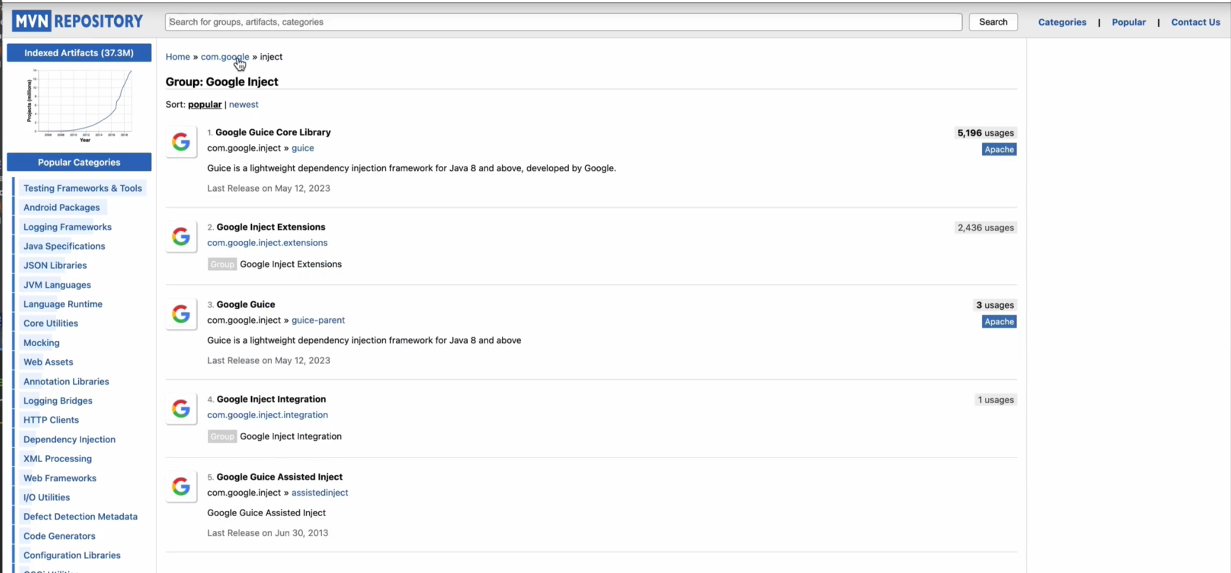


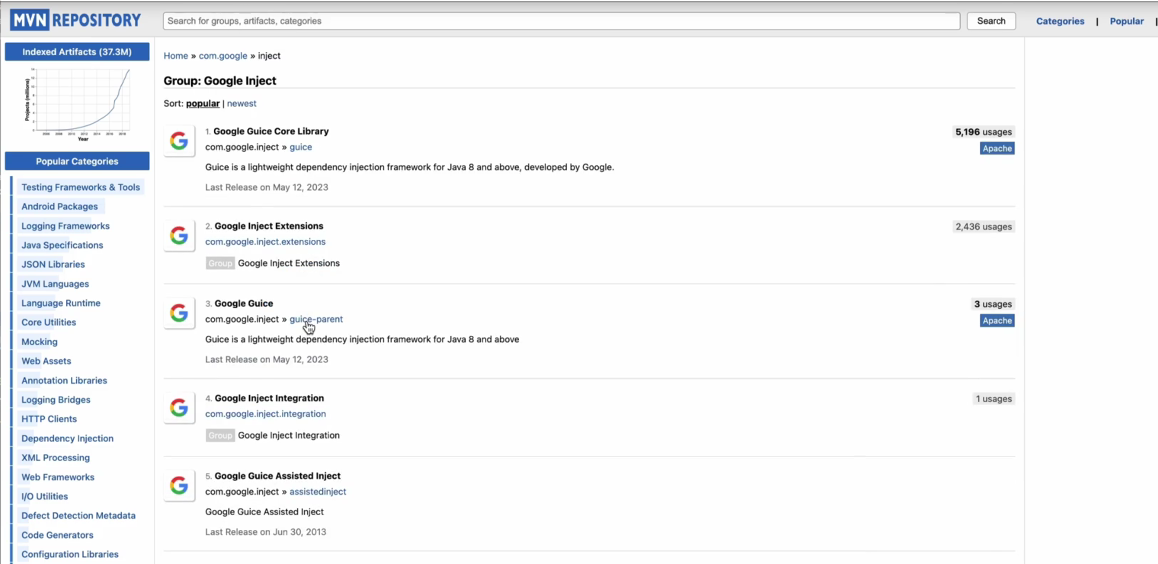
if we take a look at the POM file,

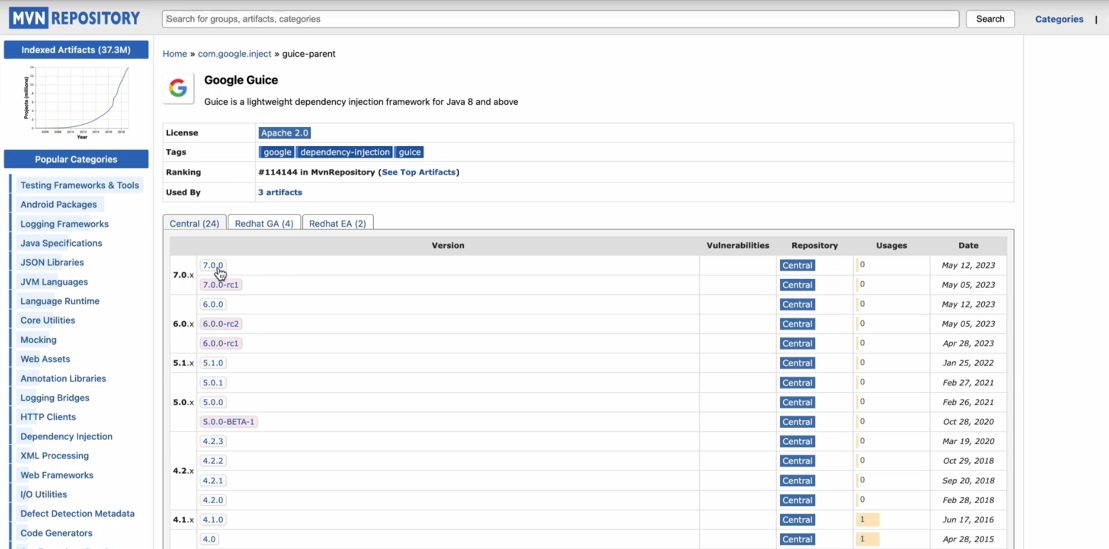


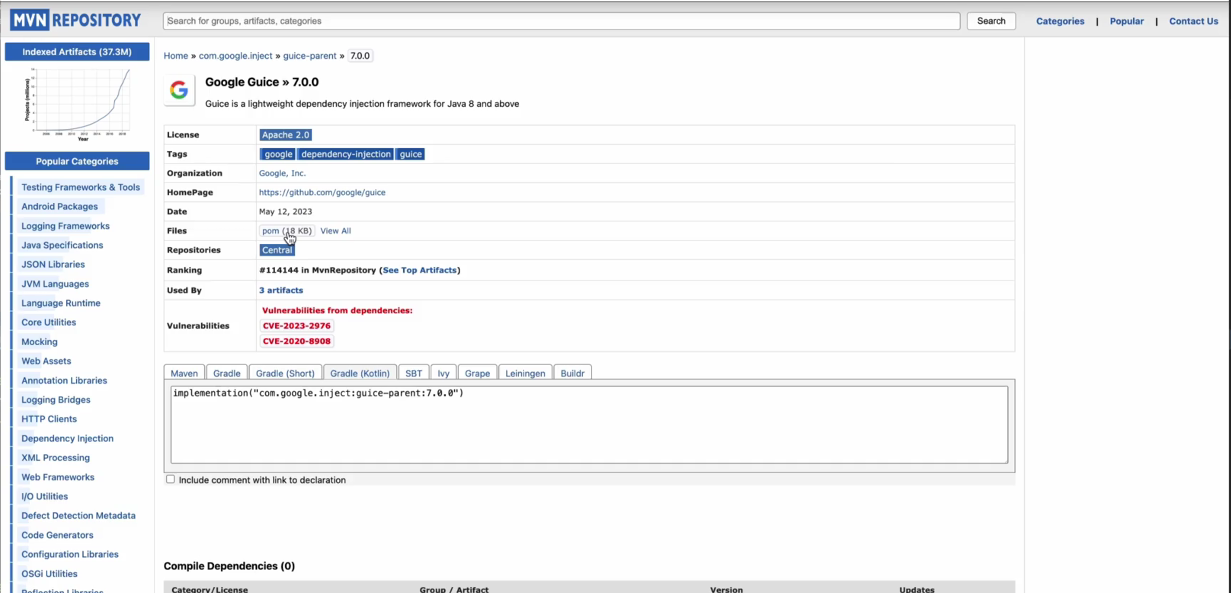


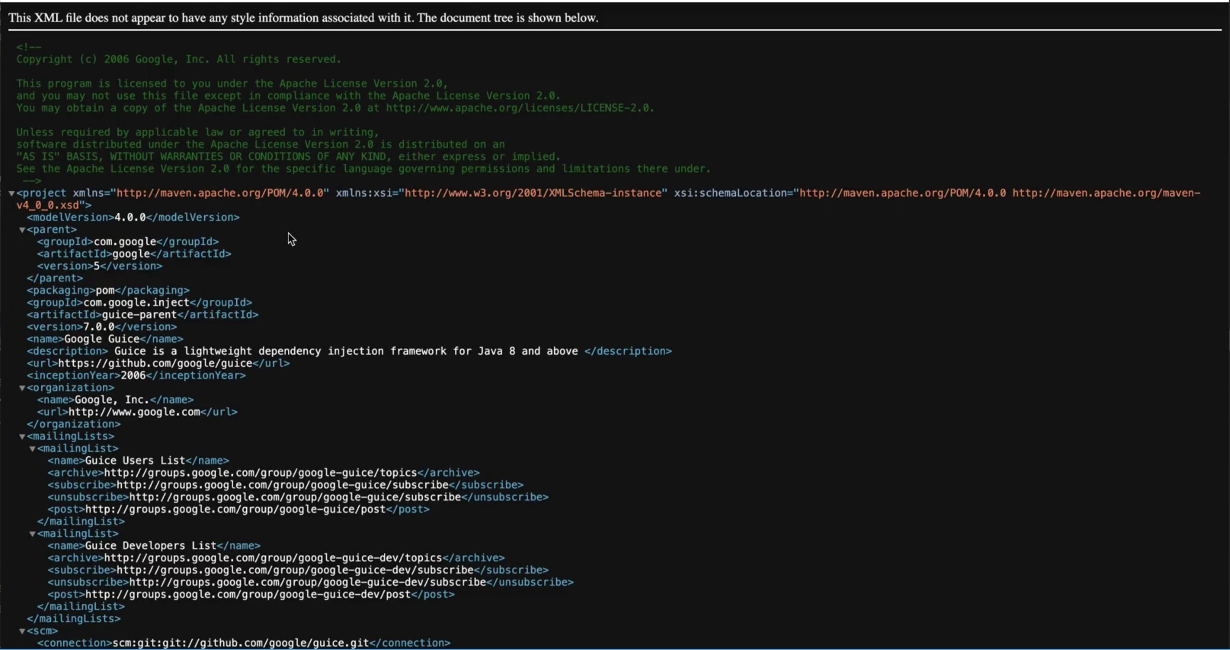
we can see the dependency here, it doesn’t have the version defined that comes from guice parent.

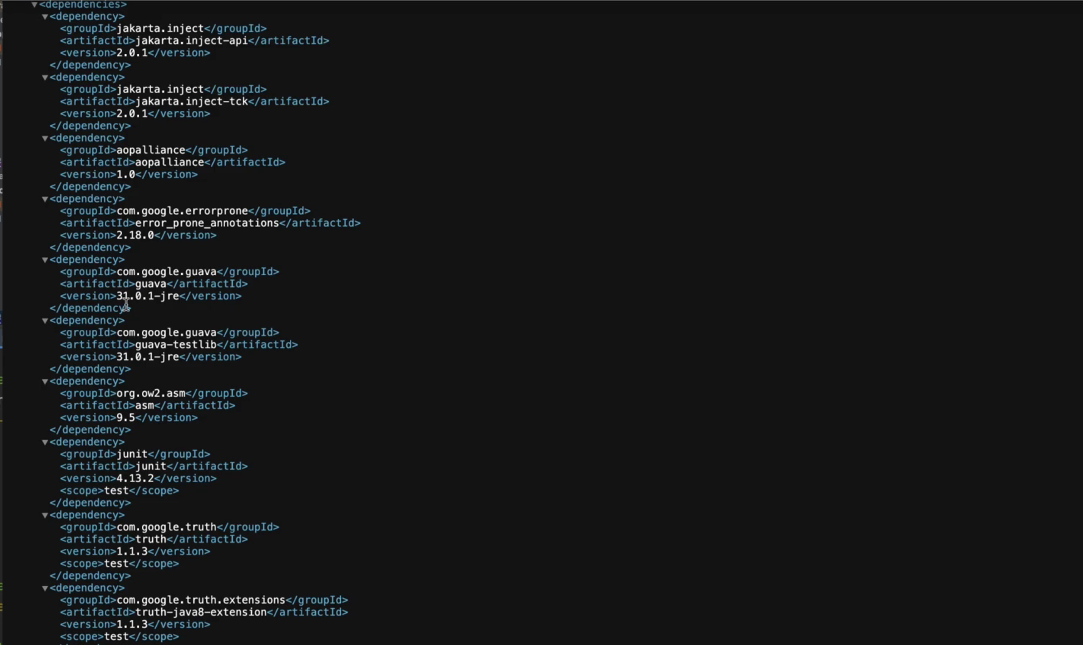




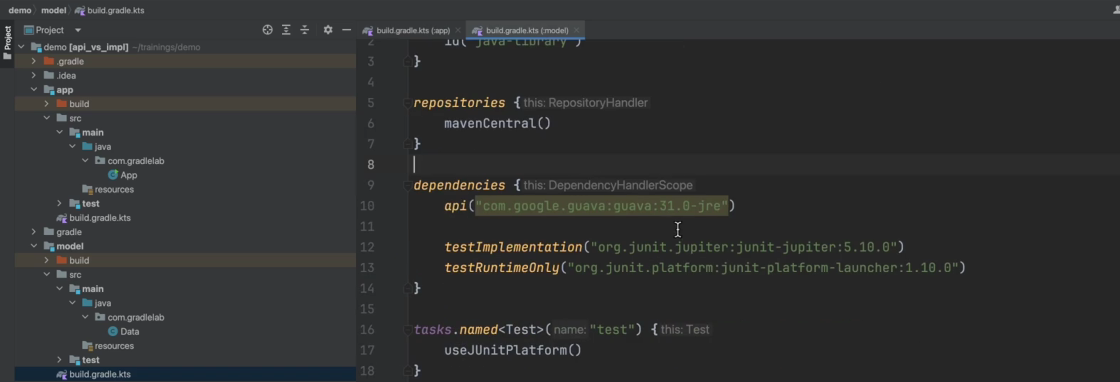






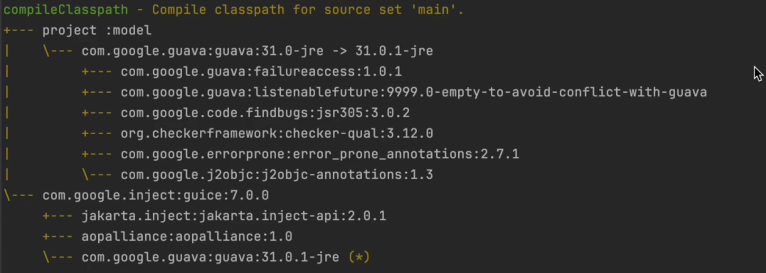


We can see it set to get at least 31.0.1-jre. That is higher than the version configured in the model subproject, which is 31.0-jre.

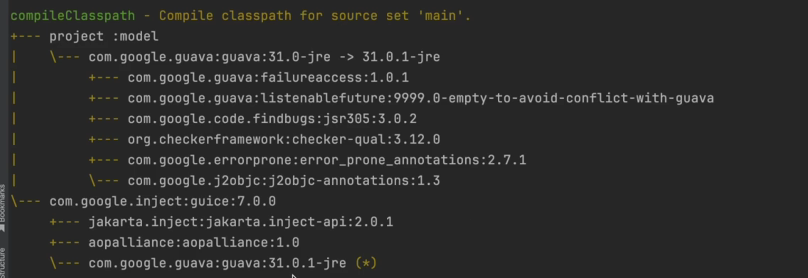


If we look at the apps dependencies now and look at the compile class path. We can see,

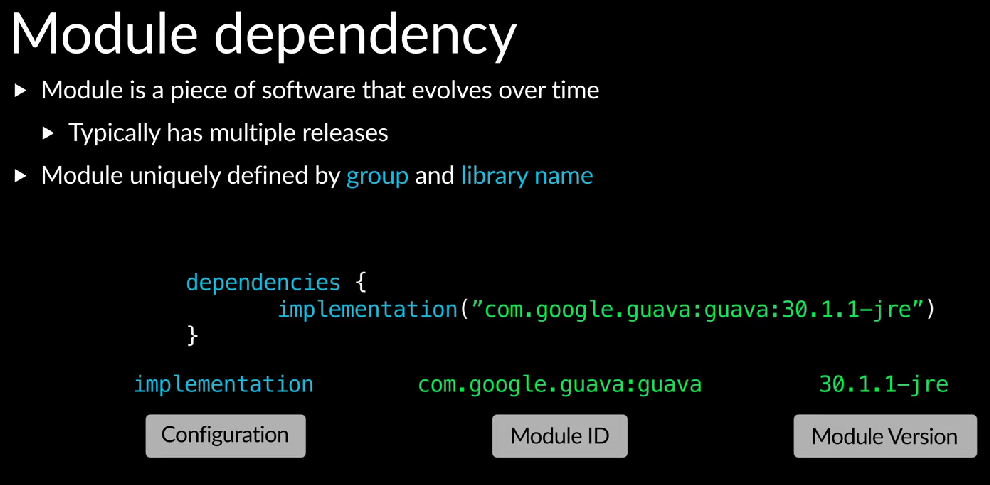




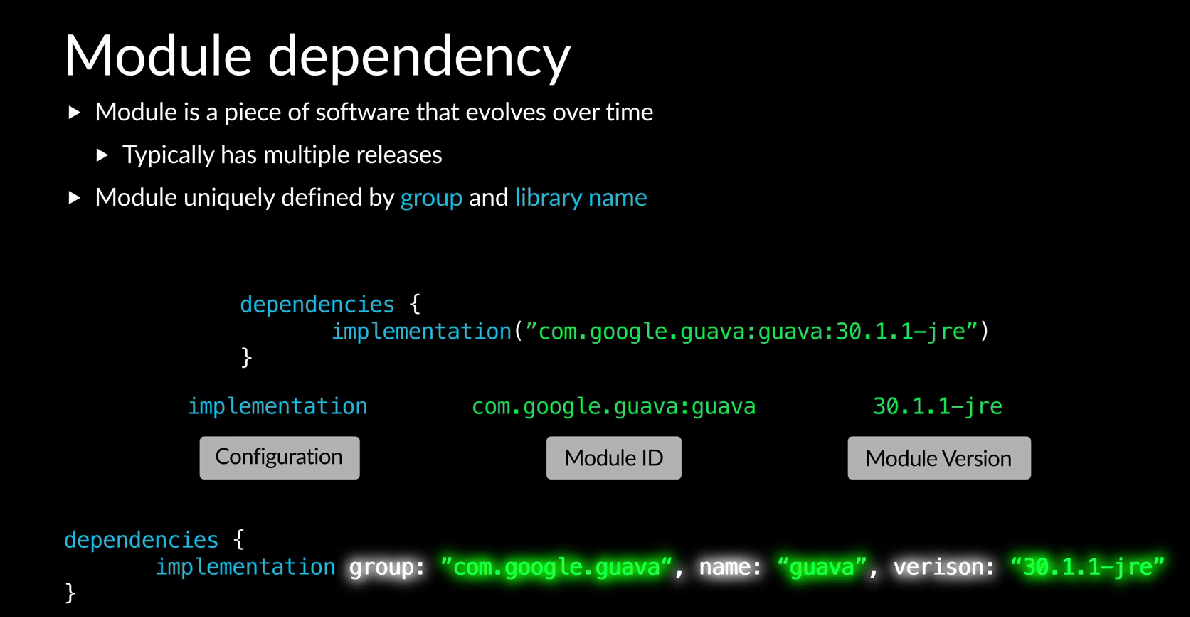
because guice is requesting a high version of guava that is,



31.0.1-jre. That higher version is used instead of 31.0-jre. If there are multiple requests of a library and of different versions only one version has to be picked. Multiple versions of a library cannot be used, so Gradle build tool has to resolve diversions and decide on one, this is known as version conflict resolution. Different build tools have different version conflict resolution strategies. In Gradle build tool, the default version conflict resolution strategy is to pick the highest version being requested. The idea behind this is simple, whoever is requesting the latest version may be using new functionality that is only there in the latest version. So picking an older version will cause issues, also, the functionality in older versions will probably still exist in the latest version, so it’s safer to pick the latest version. This does not always work since the newer version may have removed some older functionality, when that happens, you can be explicit on which version you wanna pick, and there are other strategies that can be used as well to resolve these situations which we will cover in other training.



To recap, there are three things that you need to specify when declaring a dependency, the module ID, the dependency configuration, and the version.



In projects using the Groovy DSL, you may sometimes see a longer form declaration. In Kotlin DSL, mostly the shorter form is used.

