Maven Workshop
Lab 2

1 Lab setup

Make sure you have the following items installed

- Latest version of JDK 8 / 11 (note: labs are tested with JDK 8 but should work on JDK 11 with no or minimal changes)
- Eclipse Enterprise Edition for Java (or a suitable alternative IDE for Enterprise Java)
- Latest version of Maven
- A suitable text editor (Notepad ++)
- A utility to extract zip files

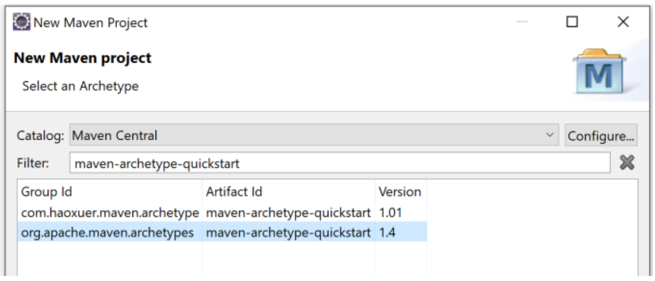
In each of the main lab folders, there are two subfolders: `changes` and `final`. The `changes` subfolder just holds the source code files for the lab, while the `final` subfolder holds the complete Eclipse project starting from its project root folder. We will use the code from the `changes` subfolder to build up our applications from scratch and you can always fall back on the complete Eclipse project if you encounter any errors while building up the application.

2 Creating a Maven project using an archetype

The source code for this lab is found in `demo-maven-basic/changes` folder.

Switch to Java EE perspective

Start with File -> New -> Maven Project. Select Next and type `maven-archetype-quickstart` in the filter. Eclipse may freeze temporarily while it attempts to filter through all the archetypes available at Maven Central. Select the entry with group id: `org.apache.maven.archetypes` and click Next


New Maven Project

New Maven project

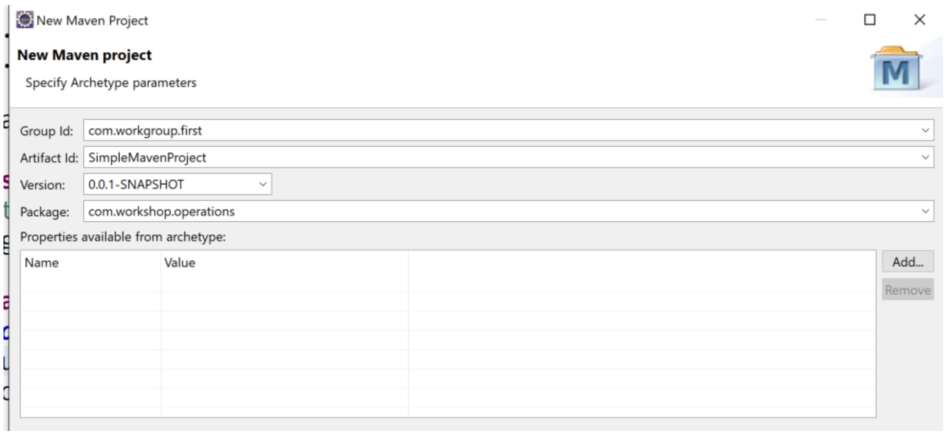
Select an Archetype

Catalog: Maven Central

Filter: maven-archetype-quickstart

Group Id	Artifact Id	Version
com.haoxuer.maven.archetype	maven-archetype-quickstart	1.01
org.apache.maven.archetypes	maven-archetype-quickstart	1.4

Enter in the following details and click Finish.


New Maven Project

New Maven project

Specify Archetype parameters

Group Id: com.workgroup.first

Artifact Id: SimpleMavenProject

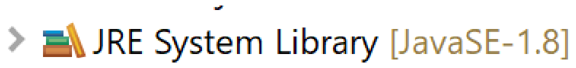
Version: 0.0.1-SNAPSHOT

Package: com.workshop.operations

Properties available from archetype:

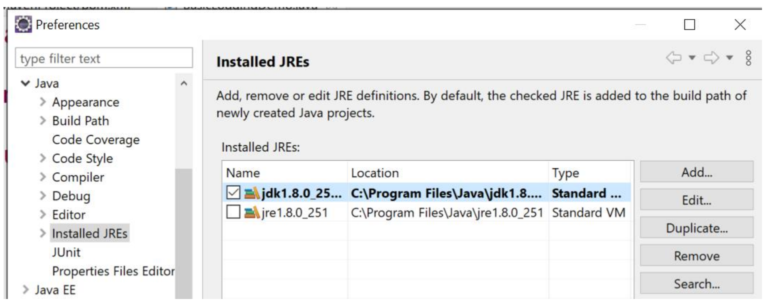
Name	Value

Replace the contents of the `pom.xml` in the project with `pom.xml` from `demo-maven-basic/changes`. Right click on the project, select Maven -> Update Project, and click OK. You should see the JRE system library entry in the project list update to JavaSE-1.8.


JRE System Library [JavaSE-1.8]

In order for a Maven build to run properly in Eclipse, you will need to use the JDK entry for the installed JRE. To check whether this is already done, go to: Window → Preferences → Java → Installed JREs

The JDK entry should be present and should be selected. If it is present and not selected, select it and click Apply and Close.


Preferences

Installed JREs

Add, remove or edit JRE definitions. By default, the checked JRE is added to the build path of newly created Java projects.

Installed JREs:

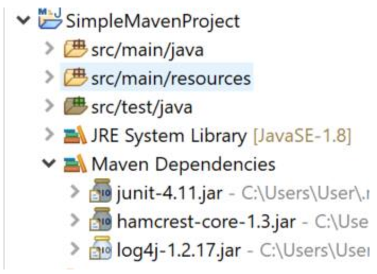
Name	Location	Type
✓ **jdk1.8.0_25...**	**C:\Program Files\Java\jdk1.8....**	**Standard ...**
jre1.8.0_251	C:\Program Files\Java\jre1.8.0_251	Standard VM

If it is not present, click Add. In the JRE Type dialog box, select Standard VM. Click Next.
In the JRE Home entry, click the Directory button and navigate and select the installation directory of the JDK, for e.g: C:\Program Files\Java\jdk1.8.0_251
Click Finish, then select the JDK entry and click Apply and Close.

If you do not do this, you will get this particular error message during the Maven build:

```
[ERROR] No compiler is provided in this environment. Perhaps you are
running on a JRE    rather than a JDK?
[INFO] 1 error
```

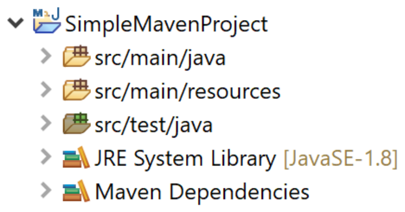
Expand on the Maven dependencies entry in the project. You should be able to see the 3 JAR files that we used in the previous lab:



```
SimpleMavenProject
  src/main/java
  src/main/resources
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
    junit-4.11.jar - C:\Users\User\.
    hamcrest-core-1.3.jar - C:\Use
    log4j-1.2.17.jar - C:\Users\User
```

Right click on project and select New -> Folder. Create a new folder named `resources` in `src/main`

Right click on project and select Properties. Select Java Build Path and click on the Source tab. Select Add Folder. Tick on the checkbox for `src/main/resources` and click ok. Click Apply and Close. You should now see this folder being registered in the entries below the project name (all these entries indicate folders which are on the application build path).



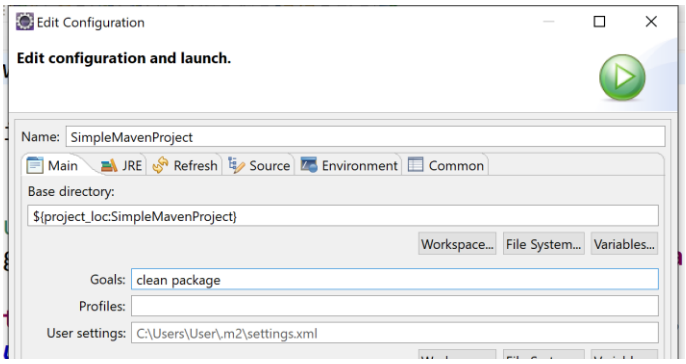
```
SimpleMavenProject
  src/main/java
  src/main/resources
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
```

Copy `BasicLoggingDemo.java` from the previous `SimpleLoggingProject` and paste it into the package `com.workshop.operations` in `src/main/java`

Copy `log4j.properties` from the previous `SimpleLoggingProject` and paste it into `src/main/resources`.

Open `BasicLoggingDemo` in the editor and do Run As -> Java application. Verify that everything works the same as before.

Right click on the project, select Run As -> 3. Maven Build. In the Edit Configuration, type the following as goals: `clean package` and click Run.



Name: SimpleMavenProject

Base directory: ${project_loc:SimpleMavenProject}

Goals: clean package

Profiles:

User settings: C:\Users\User\.m2\settings.xml

Check in the console output that the tests were run (these are JUnit tests by default) and that a JAR has been generated holding the classes from our app.

```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.workshop.operations.AppTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.013 s - in
com.workshop.operations.AppTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- maven-jar-plugin:3.0.2:jar (default-jar) @ SimpleMavenProject ---
[INFO] Building jar: G:\code\ee-eclipse\SimpleMavenProject\target\SimpleMavenProject-0.0.1-
SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.182 s
[INFO] Finished at: 2020-11-24T10:10:22+08:00
[INFO] ------------------------------------------------------------------------
```

If you run this project again as a Maven build, you will be prompted with an existing launch configuration to use. If you wish to run the Maven build with completely new goals, do Run As -> 4. Maven Build instead.

Right click on the project and select Refresh. Expand the `target` folder. Notice that it contains a JAR file in it (`SimpleMavenProject-0.0.1-SNAPSHOT.jar`).

In a separate File Explorer window, navigate to the project directory within your Eclipse workspace. Notice that the `target` subfolder actually contains two additional folders (`classes` and `test-classes`) that are not shown in the package explorer view in Eclipse. These two folders contain the compiled code from `src/main/java` and `src/test/java`

Copy the single JAR file out to another folder and check its contents with 7-Zip. Notice that

- The class files for the source code in the `src/main/java` directory are included here
- The class files for the source code in the `src/test/java` directory are NOT included here
- The `log4j.properties` file which was originally in `src/main/resources` is now in the root folder of the JAR file (i.e. corresponding to the runtime class path)

Note that this is NOT a standalone, executable JAR that you can directly run with `java -jar`. Instead, it is meant to be used as a dependency JAR for another application that needs it.

As an alternative to running Maven inside Eclipse via Eclipse's built in Maven plugin, we can also run Maven from the command line if we have Maven installed.

Open a command prompt / bash shell and navigate to the SimpleMavenProject folder in your Eclispe workspace folder and type:

```
mvn clean package
```

You will see the same console output as you did when you ran these goals using Eclipse's configuration dialog box.

We can also generate our own Maven archetype project from the command line using the `archetype:generate` plugin goal.

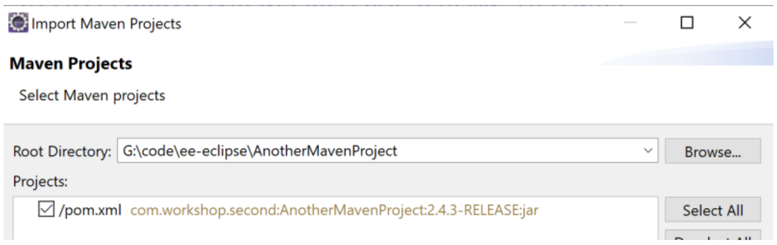
In your main Eclipse workspace directory, type the following command to generate a new Maven archetype project with specified values for the groupID, artifactID and so on passed via command line arguments prefaced with `-D`.

```
mvn      archetype:generate      -DgroupId=com.workshop.second      -
DartifactId=AnotherMavenProject       -Dversion=2.4.3-RELEASE       -
DarchetypeArtifactId=maven-archetype-quickstart    -Dpackage=jar    -
DinteractiveMode=false
```

This will create a new Maven project folder with the name `AnotherMavenProject`

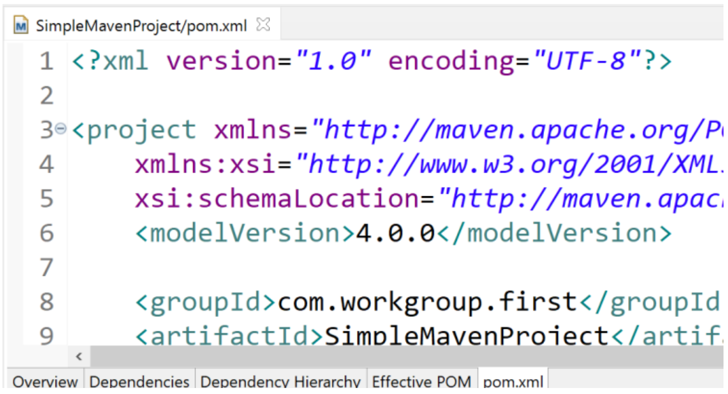
You can import this project directly into Eclipse to work on it inside there.
Do File -> Import -> Maven -> Existing Maven Projects

Click browse and look for the `AnotherMavenProject` folder, and select it and click Finish. Eclipse detects that this folder contains a `pom.xml`, and is therefore a Maven project by default.


Import Maven Projects
Maven Projects
Select Maven projects
Root Directory: G:\code\ee-eclipse\AnotherMavenProject
Projects:
☑ /pom.xml com.workshop.second:AnotherMavenProject:2.4.3-RELEASE:jar

3 Checking the POM and dependency list

Click on the tabs in the Eclipse main pane to see the 5 different representations of the POM



```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/P
    xmlns:xsi="http://www.w3.org/2001/XML
    xsi:schemaLocation="http://maven.apac
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.workgroup.first</groupId
    <artifactId>SimpleMavenProject</artif
```

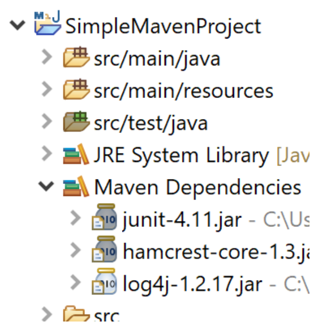
Check the `<dependencies>` section of the `pom.xml` to see where you specified the dependencies for Log4J and JUnit:

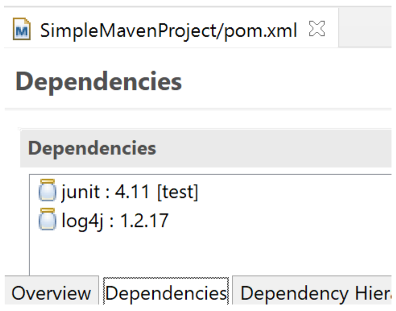
```
<dependencies>

    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
    </dependency>
</dependencies>
```

Notice that you can view the Maven dependency JARs that you have specified in your POM in the view as well as the package explorer:



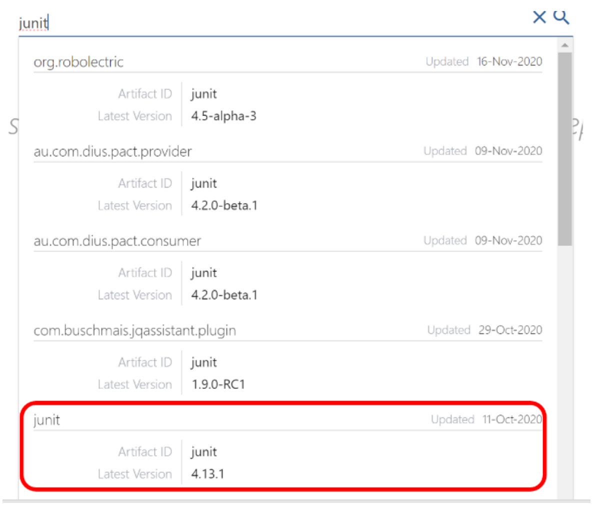


Click on the effective POM to see how it is a merging of the Super POM and your project POM
https://maven.apache.org/ref/3.6.3/maven-model-builder/super-pom.html

4 Searching for Maven dependency coordinates online

Go to the Maven Central Repository Search site : **search.maven.org**
Try to see whether you can hunt down the GAV coordinates for the two dependencies that you have included in your project: JUnit and Log4J

Type: `JUnit` in the search box. Notice that there are many `groupIDs` (representing different organizations or different depts in the same organization) for the same `artifactID`. The actual artifact that we are looking for is circled in red. Click on the Latest Version link to go there.



junit

org.robolectric		Updated 16-Nov-2020
Artifact ID	junit	
Latest Version	4.5-alpha-3	
au.com.dius.pact.provider		Updated 09-Nov-2020
Artifact ID	junit	
Latest Version	4.2.0-beta.1	
au.com.dius.pact.consumer		Updated 09-Nov-2020
Artifact ID	junit	
Latest Version	4.2.0-beta.1	
com.buschmais.jqassistant.plugin		Updated 29-Oct-2020
Artifact ID	junit	
Latest Version	1.9.0-RC1	
junit		Updated 11-Oct-2020
Artifact ID	junit	
Latest Version	4.13.1	

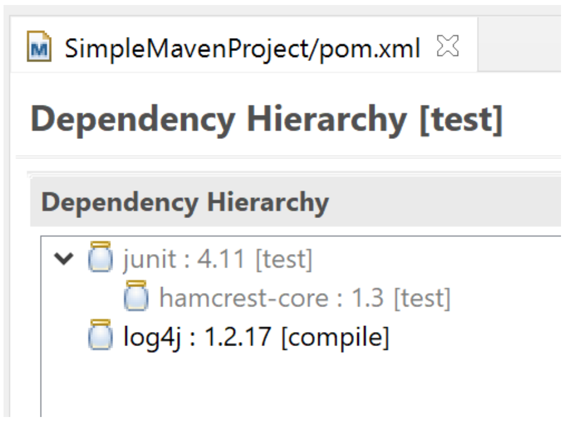
From this area, you can get the GAV coordinates for this artifact to paste in your project POM, download the JARs and also view the project POM.

If you look through the project POM for JUnit, you will see that it has a dependency of its own (Hamcrest).

```
<dependencies>
    <dependency>
        <groupId>org.hamcrest</groupId>
        <artifactId>hamcrest-core</artifactId>
        <version>${hamcrestVersion}</version>
    </dependency>

    <dependency>
        <groupId>org.hamcrest</groupId>
        <artifactId>hamcrest-library</artifactId>
        <version>${hamcrestVersion}</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

However, in your own POM, you don't have to specify this dependency explicitly. Instead Maven is intelligent enough to figure out this additional dependency on its own (called transitive dependency) when it downloads the POM for JUnit. It will then additionally fetch the Hamcrest dependency as well. You can verify this in the Dependency Hierarchy view perspective of the POM.xml

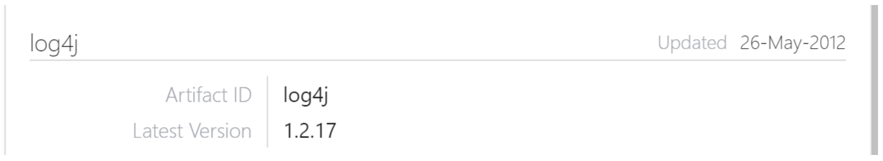

SimpleMavenProject/pom.xml

Dependency Hierarchy [test]

Dependency Hierarchy

```
junit : 4.11 [test]
    hamcrest-core : 1.3 [test]
log4j : 1.2.17 [compile]
```

Repeat this for log4j. Note that the artifact we are actually looking for is the older version of log4j and not the newer version log4j 2.


log4j   Updated 26-May-2012

Artifact ID	log4j
Latest Version	1.2.17

If you want a more detailed categorization of your search items, use the Classic Search menu instead:

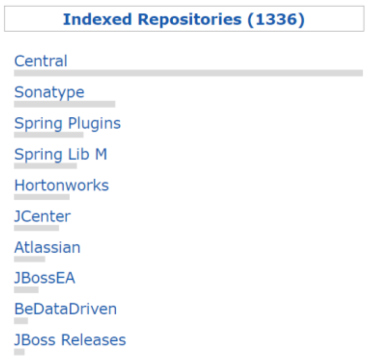
https://search.maven.org/classic/

Click on Advanced Search.

One of the more useful things to search for is based on classname. So for e.g. if you have a piece of source code that you copied from a website which uses external Java libraries, but you have no idea which library it uses (and therefore cannot search for it properly), you can search on the imported classes used in that source code. You may get a lot of results which you can browse through to attempt to find the correct organization and artifact.

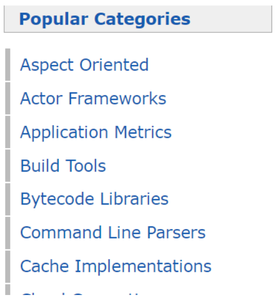
In addition to the Maven Central Repository, most organizations also host publicly accessible Maven repositories which may contain dependencies that are not available in the Maven Central repository. The site below indexes all these additional repositories
https://mvnrepository.com

You can see a short clip of the top indexed repositories on the right-hand of the main page. You can see The Maven Central Repository is right there at the top. Notice that the Spring project itself has two public repositories (Spring Plugins and Spring Lib M) which have an extremely high number of indexed JARs. This goes to show how active the Spring framework community is !


Indexed Repositories (1336)

Central
Sonatype
Spring Plugins
Spring Lib M
Hortonworks
JCenter
Atlassian
JBossEA
BeDataDriven
JBoss Releases

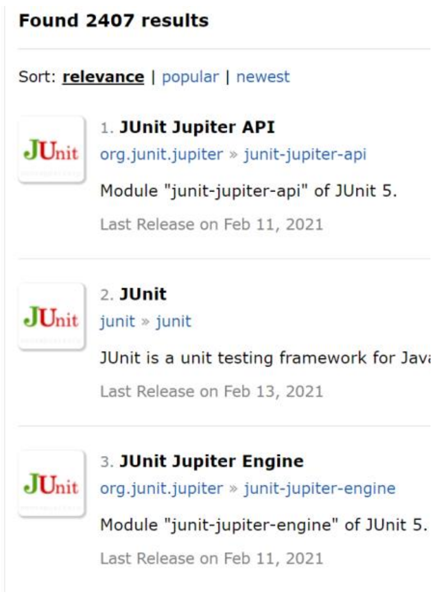
The Popular Categories list on the left hand side allows you to quickly determine the most popular projects for a particular type of usage category. This can be very helpful in determining which particular Java framework/library that you want to use in the event there are several candidates possible.


Popular Categories

Aspect Oriented
Actor Frameworks
Application Metrics
Build Tools
Bytecode Libraries
Command Line Parsers
Cache Implementations

You can also do a search here for the artifact that you are looking for here as well, and the search results are generally more easy to understand than at **search.maven.org**

For e.g. typing `junit` in the search box returns a list of projects listed by their group and artifact IDs, from which you can drill down further by clicking on either one of these IDs:


Found 2407 results

Sort: relevance | popular | newest

1. JUnit Jupiter API
org.junit.jupiter » junit-jupiter-api
Module "junit-jupiter-api" of JUnit 5.
Last Release on Feb 11, 2021

2. JUnit
junit » junit
JUnit is a unit testing framework for Java
Last Release on Feb 13, 2021

3. JUnit Jupiter Engine
org.junit.jupiter » junit-jupiter-engine
Module "junit-jupiter-engine" of JUnit 5.
Last Release on Feb 11, 2021

Clicking on a specific artifact ID gives you the list of artifact versions at different repos, along with their usage statistics. Going to specific version page allows you obtain the corresponding GAV coordinates and download the JAR files.

Another way to determine the Maven GAV coordinates for a dependency that you need to use in your project is to search through the home website dedicated to that project (most major Java libraries have a dedicated home website)

For e.g. if we want to use Hibernate (a popular JPA ORM provider), we can find its Maven dependency here
https://hibernate.org/orm/documentation/getting-started/

If we want to use Logback (another popular Java logging framework), we can find its Maven dependency here
http://logback.qos.ch/setup.html

If you can't find this on the project home website, you can try doing a search on:

```
xxxx maven dependencies
```

in google search, where *xxx* is the name of the artifact/project you are looking for. Usually the first 5 – 10 links will either navigate you to https://mvnrepository.com/, from which you can then explore further. Or it might redirect you to another website which lists the dependencies that you are looking for.

For e.g. if I type:

```
spring maven dependencies
```

I get links to the following sites that can help me locate the Maven dependency coordinates for the specific artifact I am interested in:

https://mvnrepository.com/artifact/org.springframework/spring-core

Alternatively this article, gives you information on the dependencies for all the major components of the Spring framework
https://www.baeldung.com/spring-with-maven

5 Online and local repositories

The central Maven repository where the actual dependency JARs for the various project artifacts that you need to download and use in your project at is located at these links:

(newer): https://repo.maven.apache.org/maven2
(older): https://repo1.maven.org/maven2/

Go here and navigate down through any one of the project links to locate the various JAR files and other project artifacts (source code, documentation, project POM, etc)

Go to the default location for the local Maven repository cache on your machine:

Windows: `C:\Users\<User_Name>\.m2\repository`
Linux: `/home/<User_Name>/.m2/repository`
Mac: `/Users/<user_name>/.m2/repository`

Check that you already have a folder structure corresponding to the GAV coordinates of the two dependencies that you downloaded, for e.g:

```
C:\Users\<User_Name>\.m2\repository\junit\junit\4.11
C:\Users\<User_Name>\.m2\repository\log4j\log4j\1.2.17
```

Notice that the folder contains the JAR, source code as well as project POM. The next time your run a build that references these dependencies, Maven will first check in the local repo cache and use these JARs first. It will only check the central Maven repository if it can't find it there.

Try and delete any of these folders and then do a Maven -> Update Project on SimpleMavenProject. You will see Maven download all of the relevant dependency artifacts and create the folder anew to place them within it.