Maven Workshop

Topics

[Prerequisites 2](#_Toc402309746)

[Configuring Maven 2](#_Toc402309747)

[Create and run Maven project 3](#_Toc402309748)

[Using Dependencies 7](#_Toc402309749)

[Unit Testing 9](#_Toc402309750)

[Maven Plugins 10](#_Toc402309751)

[Local repository 11](#_Toc402309752)

[Documentation and Reporting Plugins 12](#_Toc402309753)

# Prerequisites

1. IntelliJ IDEA
2. JDK 8
3. Maven 3
4. Tortoise Git or other GIT client

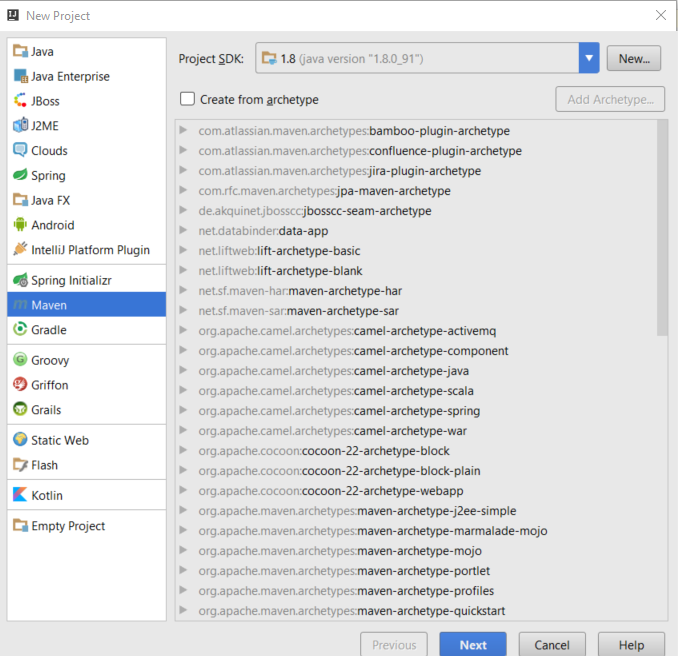
# Configuring Maven

* Extract the archive with the Maven distribution
* Create a user variable **M2\_HOME** and set the value to your root maven folder (Ex: D:\Tools\apache-maven-3.2.3)
* Add the bin directory to you Path Environment variable (Ex: %M2\_HOME%\bin)
* Open a cmd window and verify that Maven is set correctly by running **mvn –version** command.
* Check if JAVA\_HOME variable exists, if not, create a user variable JAVA\_HOME and set the value to your root java folder (Ex: C:\Program Files\Java\jdk1.8.0\_91)

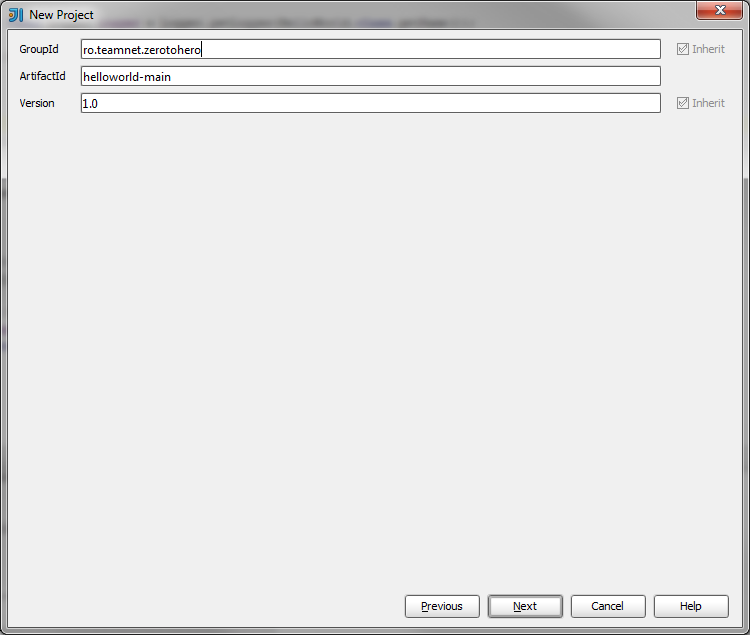
# Create and run Maven project

Create a new Maven project with IDEA.

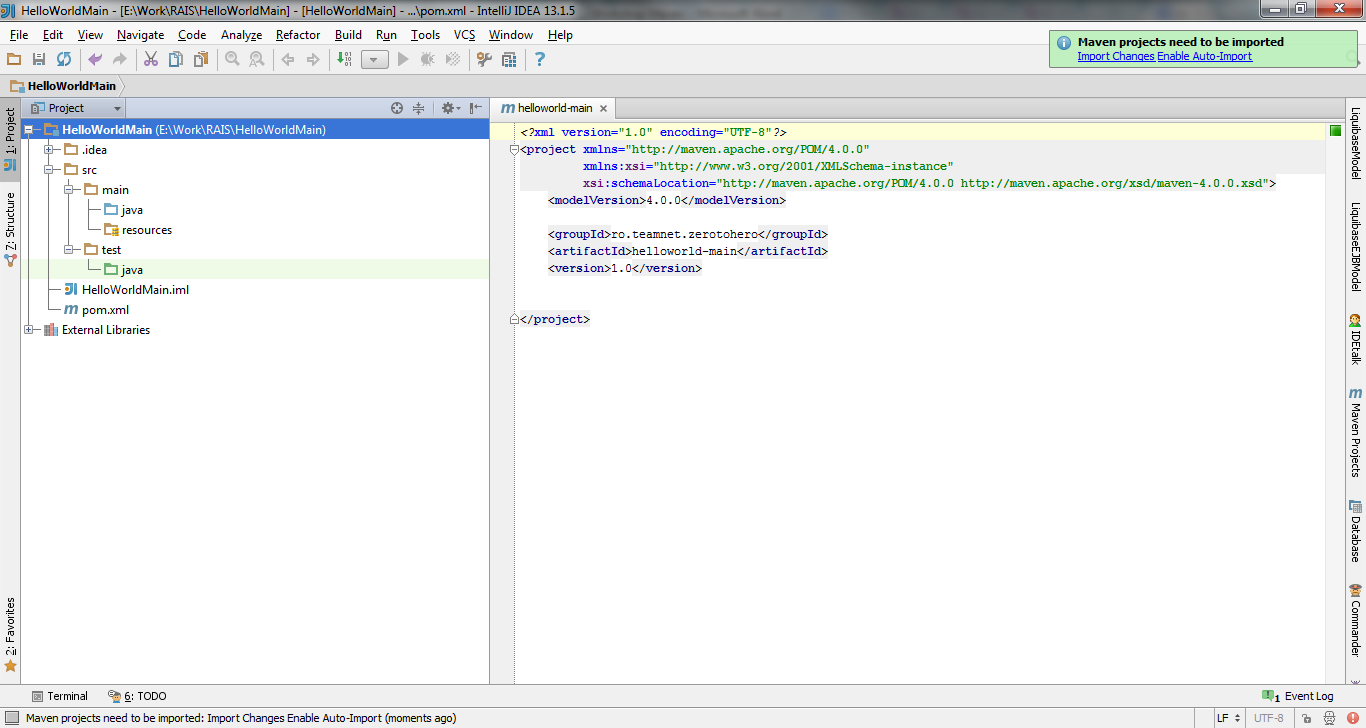
1. Open IntelliJ IDEA ☺
2. File -> New -> Project
3. Select ***Maven*** from the left menu
4. Select 1.8 (or desired java version) to **Project SDK** option
5. Click **Next**



1. Set the following values to the next form.
   1. GroupId to **ro.teamnet.zerotohero**.
   2. Artifact to **helloworld-main**
   3. Version to **1.0**



1. In the next window:
   1. Enter **HelloWorldMain** value to the ***Project name*** field
   2. Set your **Project Location** inside your Java git Branch folder (<local\_git\_path>/maven/exercises/{HelloWorldMain})
   3. Click Finish.
2. You should have a project with the below format:



1. Adding a java class file:
   1. Under src/main/java create a new package **ro.teamnet.hello**
   2. In **ro.teamnet.hello** create a new class named **HelloWorld**
   3. Add a method sayHello() and one returnHelloKey() as below:

**/\*\***

**\* method for saying hello**

**\*/**

**public void sayHello(){**

**System.out.println("Hello World!");**

**}**

**/\*\***

**\* method for returning a key**

**\* @return - The HelloWorld key**

**\*/**

**public String returnHelloKey(){**

**return "HelloKey";**

**}**

1. Go into pom.xml and add following lines after <version>

**<description>A Maven project for displaying a Hello World Application</description>**

**<name>Hello World Project</name>**

1. Create a main method for the class HelloWorld and run it.

**public static void main(String[] args) {**

**HelloWorld helloWorld = new HelloWorld();**

**helloWorld.sayHello();**

**}**

Guess what: It will print HelloWorld! ☺

1. Run the install task from Intellij (go to View and check Tool Buttons if it’s not checked). You will see the Maven Button. Expand your project and go to Lifecycle -> install and double click on install. This will invoke the install phase. After install is completed, go into your root project folder. You should find the target folder. There will be the generated jar of your application.
2. Here we can also play with some other tasks:

**clean** – it will clean the target and all the generated .class files

**validate** – for this you can comment line **<groupId>ro.teamnet.zerotohero</groupId>** from pom.xml

**package –** creating the package (in our case the Jar)

# Using Dependencies

1. Now let’s place a Logger plugin so that we can add logs into our code. We will use Log4J (<http://logging.apache.org/log4j/1.2/> ).
2. Create a file named log4j.properties under src/main/resources and add the following code into it:

**log4j.rootLogger=debug, stdout, R**

**log4j.appender.stdout=org.apache.log4j.ConsoleAppender**

**log4j.appender.stdout.layout=org.apache.log4j.PatternLayout**

**# Pattern to output the caller's file name and line number.**

**log4j.appender.stdout.layout.ConversionPattern=%5p [%t] (%F:%L) - %m%n**

**log4j.appender.R=org.apache.log4j.RollingFileAppender**

**log4j.appender.R.File=logs.log**

**log4j.appender.R.MaxFileSize=100KB**

**# Keep one backup file**

**log4j.appender.R.MaxBackupIndex=1**

**log4j.appender.R.layout=org.apache.log4j.PatternLayout**

**log4j.appender.R.layout.ConversionPattern=%p %t %c - %m%n**

1. So that we can use the Log4J API we need to add it as a dependency to our pom.xml

**Let’s rearch for the dependency inside the default maven repository (if you reached these step please stop here)**

1. We will use the dependency in the pom.xml file inside the <dependencies> tag:

<dependencies>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

</dependencies>

1. Define the log4j Logger for the HelloWorld class:

**static final Logger logger = Logger.getLogger(HelloWorld.class.getName());**

1. Add the following 2 lines into the sayHello() method:

**logger.debug("DEBUG -> Enters in sayHello() method from HelloWorld");**

**logger.info("INFO -> Enters in returnHelloKey from HelloWorld");**

1. Run the main method again. You will see 3 lines there:

**Hello World!**

**DEBUG [main] (HelloWorld.java:21) - DEBUG -> Enters in sayHello() method from HelloWorld**

**INFO [main] (HelloWorld.java:22) - INFO -> Enters in returnHelloKey from HelloWorld**

1. Go into log4j.properties and modify first line as below:

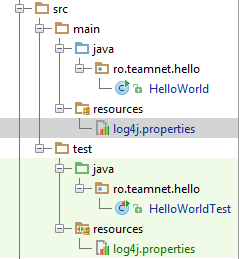
**log4j.rootLogger=info, stdout, R**

Now run the main method again

All these output can also be viewed in the **logs.log** file in the root folder of your project.

# Unit Testing

1. Get the JUnit maven dependency (version 4.12) from mvn repository and add it to your pom.xml
   1. Go to <https://mvnrepository.com>
   2. Search for “junit”
   3. Enter on the first result
   4. Click on the latest version (should be [4.12](https://mvnrepository.com/artifact/junit/junit/4.12))
   5. Copy the content from the **Maven** tab and paste it in your pom.xml file to the <dependencies> tag
2. Create a class **HelloWorldTest** under **src/test/java** in the package **ro.teamnet.hello** (also create the package)
3. Create a new directory **resources** under **src/test**



1. Create 2 methods in the class HelloWorldTest and annotate them with the @Test annotation:

@Test

public void testSayHello(){

HelloWorld helloWorld = new HelloWorld();

helloWorld.sayHello();

}

@Test

public void testReturnHelloKey(){

HelloWorld helloWorld = new HelloWorld();

System.out.println("From Test: " + helloWorld.returnHelloKey());

}

1. Right click on each method and click Run. This will run the Unit Test.
2. Add the following line to the testReturnHelloKey() method:

**assert helloWorld.returnHelloKey().equals("HelloKey");**

Run the test again

1. Change “HelloKey” to “Hello Key” in the above line. Run the test again. The test will fail.

# Maven Plugins

1. Copy the following plugin inside pom.xml:

**<build>**

**<plugins>**

**<plugin>**

**<groupId>org.apache.maven.plugins</groupId>**

**<artifactId>maven-source-plugin</artifactId>**

**<executions>**

**<execution>**

**<id>attach-sources</id>**

**<goals>**

**<goal>jar</goal>**

**</goals>**

**</execution>**

**</executions>**

**</plugin>**

**</plugins>**

**</build>**

1. Run the install again and check the target folder. It will also contain the Java sources package and it will also be added to the local repository.

# Local repository

1. Create a new Maven Project
   1. GroupId to **ro.teamnet.zerotohero**.
   2. Artifact to **helloworld-extend**
   3. Version to **1.0**

**Project Name: HelloWorldExtend**

1. Add in pom.xml the dependency to the first project library
2. Similarly to first project, create a class HelloWorldExtend in the package ro.teamnet.hello2:

**public class HelloWorldExtend {**

**public HelloWorldExtend() {**

**}**

**public void extendSayHello(){**

**HelloWorld helloWorld = new HelloWorld();**

**helloWorld.sayHello();**

**System.out.println("The new Hello World");**

**}**

**}**

1. Add a new test class to the project, named **HelloWorldExtendTest** and create a new method in wich you will call the **extendSayHello()** method.

Adauga pachetul ro.teamnet.hello2 in scr/test/java in care adugam clasa:

**public class** HelloWorldExtendTest {  
 @Test  
 **public void** test() **throws** Exception {  
 **new** HelloWorldExtend().extendSayHello();  
 }  
}

# Documentation and Reporting Plugins

Maven **site** helps us build our project documentation very efficiently and brings a lot of plugins around this area so that someone can generate automatically a complete project documentation without having to write it in a document.

All you have to do is WATCH your code by:

* Adding Java Docs comments on every class and method explaining exactly their purpose and usage
* Add Unit tests as much as possible so that the code can be very well tested
* Complete your pom.xml with all infos regarding the team that developed the project, other contributors, mailing lists project description and so on.

The example will be on branch petrusprangate in the Java repository and you can play with the pom.xml from there to see a lot of excited features.

In the workshop a part of them will be presented just as a usage scenario.