

Spring Core Unit – Software Development Lifecycle

Lesson 2: Intro to Maven



SOFTWARE-GUILD

Copyright © 2016 The Learning House.

All rights reserved. No part of these materials may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of The Learning House. For permission requests, write to The Learning House, addressed "Attention: Permissions Coordinator," at the address below.

The Learning House
427 S 4th Street #300
Louisville KY 40202

Lesson 2: Intro to Maven

Overview

So far, we have been using the build management tools that are built into NetBeans. This works reasonably well for small, individual projects that don't depend on too many external libraries, but it has its drawbacks for larger, multi-developer projects:

Everyone on the team has to use NetBeans.

All external jar files must be referenced directly in the project and must be manually copied to each developer's machine.

There is no good way to manage the versions of the required external libraries.

Maven helps to address these issues. Maven is a project management framework that provides IDE independent build and dependency management tools.

What is Maven?

Maven bills itself as a "Project Management Framework" — it strives to manage a project's build, reporting, and documentation from one place.

Maven's build management is declarative rather than task-oriented. It has a built-in lifecycle so you simply declare what you want to do — not how to do it. Maven also has declarative dependency management, so you tell Maven what libraries (including version numbers) you need and it will make sure those libraries are available to your code.

Maven does a lot of things and is a pretty big subject. In this course, we will concentrate on using Maven, the built-in Maven lifecycle, and the dependency management features to make our projects easier to build and to share with our team.

Project Object Model

Maven is based on the Project Object Model or POM. The POM is defined in an XML file called pom.xml. This file contains the declarations for all libraries on which the project depends and can contain declarations of which Java version to use and other project level settings. Listing 1 shows a typical POM file. This particular example relies on the JUnit 3.8.1 jar and specifies that Java 8 (i.e. 1.8) should be used to both compile and run this project.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.swcguild</groupId>
  <artifactId>MeanMedianMode</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>MeanMedianMode</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <finalName>mean-median-mode</finalName>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.3.1</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

Listing 1 - Maven pom.xml File

Maven automatically fetches all dependencies into a central repository on your machine (located in the `~/.m2` directory). All Maven projects on your machine share this repository, which means that each library is only downloaded once. As an added bonus, Maven also handles **transitive dependencies** automatically. For example, if you declare (in your POM) that your project depends on Library A and it turns out that Library A, in turn, depends on Libraries B and C, Maven will automatically download all three libraries into your local repository — you do not have to specify (or even be aware) that Library A requires Libraries B and C.

Maven Lifecycle

Maven's project lifecycle is defined but is flexible — you can change it if you need to but, for most projects, the predefined lifecycle is sufficient. The lifecycle consists of several stages which are known as **goals**. We will be using the following goals extensively:

1. `compile`: compiles the project source code
2. `test-compile`: compiles the project test source code
3. `test`: runs the project unit tests
4. `package`: builds and packages the project
5. `install`: installs the project package into the local `.m2` repository (the project package can then be used in other projects)

Profiles

The Profile feature allows you to customize settings for particular target environments (e.g. to set up different database connection settings for dev, test, and production). Enabling this feature is a three-step process:

1. Add a profile tag to the POM file. This tag will contain the values that should be substituted into the configuration files per target environment.
2. Edit target configuration files by replacing actual configuration settings with placeholder tokens that will be replaced by Maven.
3. Enable filtering. This causes Maven to replace the tokens in the configuration files with actual values from the profile tag.

We won't use this feature extensively, but we will touch on it later in the course.

Installation and Tour

NetBean comes with an internal version of Maven that it can use. This version works well but it is not the latest version of Maven and it cannot be run from the command line. Because of this, we will install the latest version of Maven and then configure NetBeans to use our version rather than its internal version.

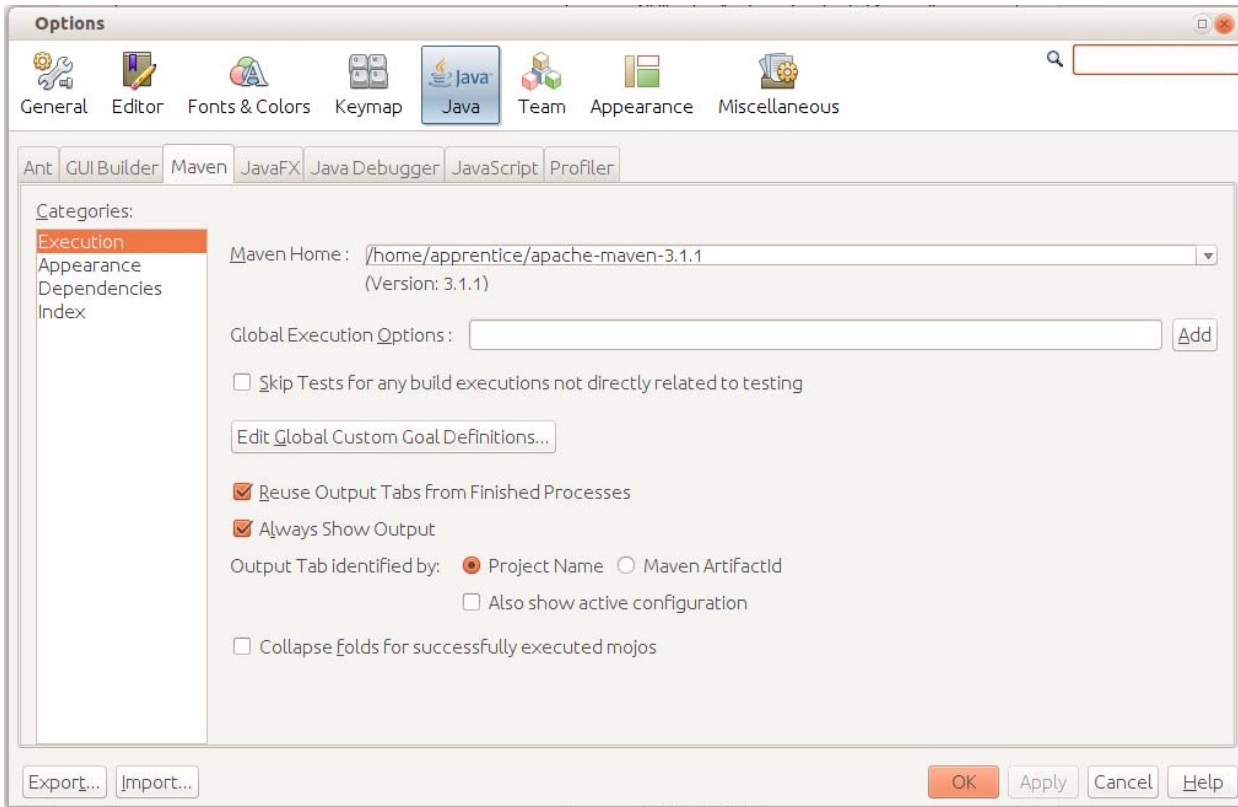
Download and Install Maven

Go to <http://maven.apache.org/download.cgi> and download the binary (bin.tar.gz) tarball for the latest version of Maven. Unzip the file and follow the directions found in README.txt.

Configure NetBeans

Now, we must configure NetBeans to use the external Maven instance.

First, from the Tools menu, select Options. In the Options dialog, select “Java” as the main category and then go to the “Maven” tab:

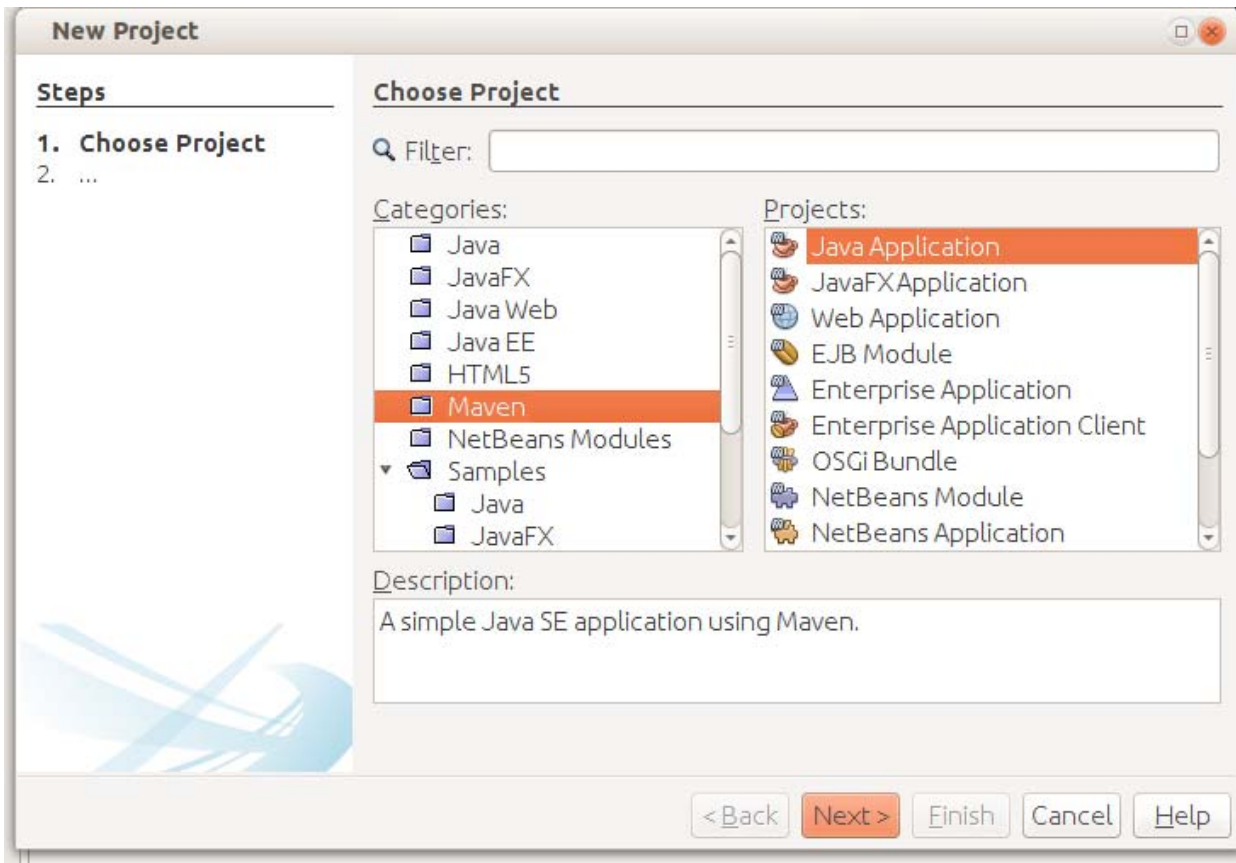


Next, click the Maven Home drop down menu, select “Browse,” and then browse to the folder in which you installed Maven. When done, it should look similar to the image above. Finally, click OK. NetBeans is now configured to use an external instance of Maven.

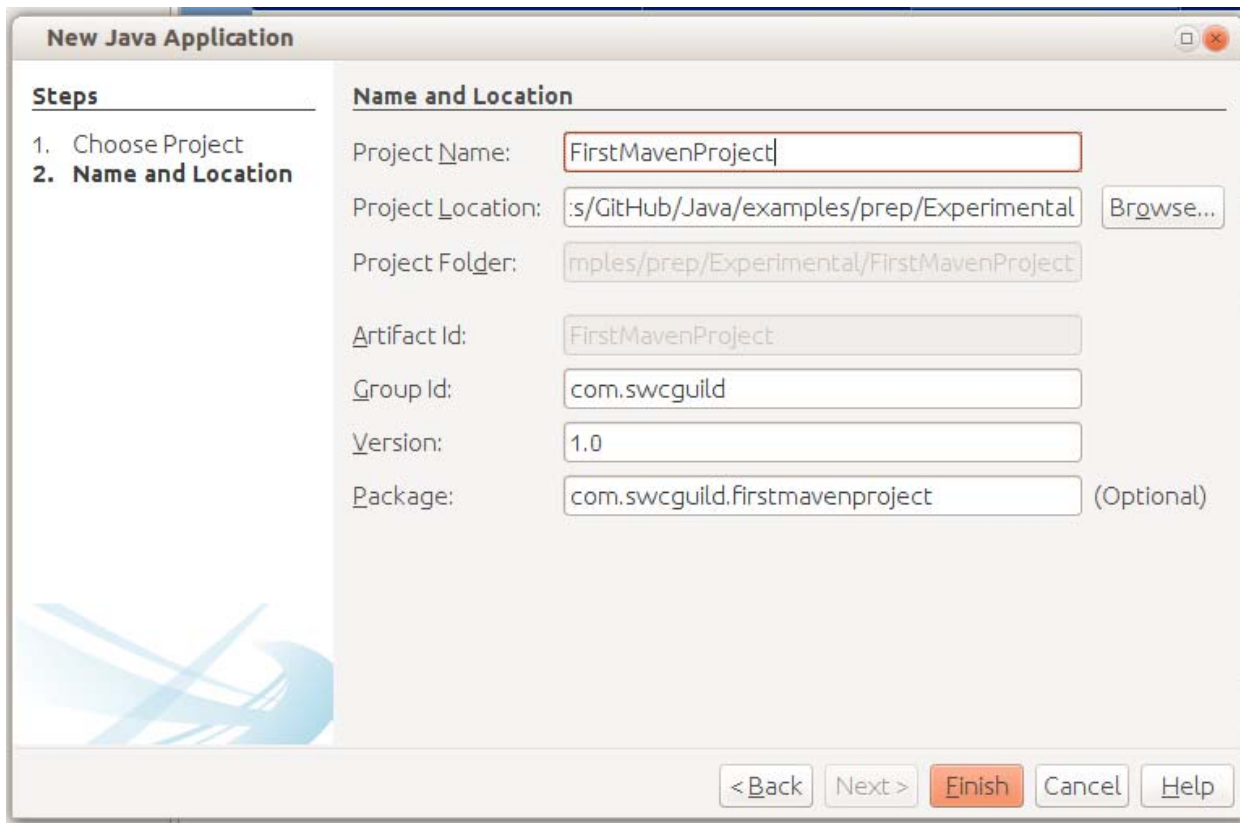
Create a Maven Project

From this point on, we will create all of our projects as Maven projects rather than NetBeans projects. Both the steps to create a Maven project and the resulting project structure are slightly different than the standard NetBeans project.

First, from the File menu, select New Project. In the New Project dialog, select Maven (under Categories) and Java Application (under Projects), and click Next:



In the New Java Project dialog, type in the name of your project, select a Project Location, fill in the Group Id, and finally supply a version number:



New Java Application

Steps

1. Choose Project
2. **Name and Location**

Name and Location

Project Name: FirstMavenProject

Project Location: s:/GitHub/Java/examples/prep/Experimental

Project Folder: examples/prep/Experimental/FirstMavenProject

Artifact Id: FirstMavenProject

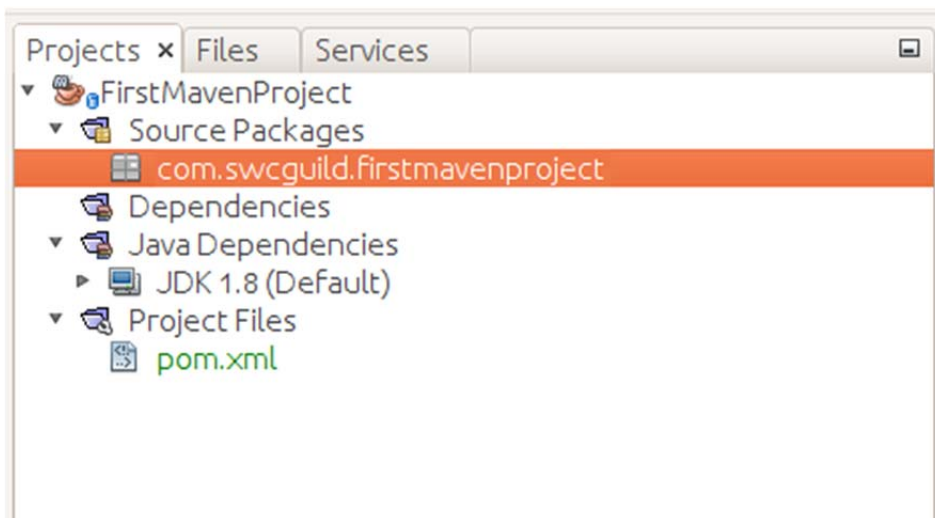
Group Id: com.swcguild

Version: 1.0

Package: com.swcguild.firstmavenproject (Optional)

< Back Next > **Finish** Cancel Help

The project will be created when you click the Finish button. The project structure should look similar to the following:



Customization

We will not be customizing Maven beyond what has been outlined above. However, there are a couple of common customizations that people and organizations apply to Maven that you should be aware of as you enter the industry:

1. Local mirror: many companies host their own Maven repository because it allows them to control exactly what dependencies can and cannot be declared. In this configuration, all developers configure their local Maven installation to get all dependencies from an internal company server. If the required dependency doesn't exist in the local mirror, the developer must formally request that it be added to the local mirror.
2. Some developers change the location of the .m2 repository on their local machines. One reason to do this is if they use multiple drives on and want the .m2 repository on a specific drive because of space constraints.