Maven

What is Maven?

Central concept of Maven is a ‘build lifecycle’ == “the process for building and distributing a particular artifact (project) is clearly defined”

There are three build lifecycles:
‘default’ – handles project deployment
‘clean’ – handles project cleaning------
‘site’ – handles creation of projects site documentation

Each lifecycle is made up of phases which are executed sequentialy:

- validate - validate the project is correct and all necessary information is available
- compile - compile the source code of the project
- test - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
- package - take the compiled code and package it in its distributable format, such as a JAR.
- integration-test - process and deploy the package if necessary into an environment where integration tests can be run
- verify - run any checks to verify the package is valid and meets quality criteria
- install - install the package into the local repository, for use as a dependency in other projects locally
- deploy - done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

Furthermore, each phase is made up of specific goals which must be completed for the phase to succeed. some goals can be used in multiple phases

Maven uses a P.O.M. (Project Object Model) which is an XML representation of the Maven project. The POM contains all necessary information about a project, as well as configurations of plugins to be used during the build process.

Here is the basic POM XML file structure

<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>

<!-- The Basics -->

<groupId>...</groupId>

<artifactId>...</artifactId>

<version>...</version>

<packaging>...</packaging>

<dependencies>...</dependencies>

<parent>...</parent>

<dependencyManagement>...</dependencyManagement>

<modules>...</modules>

<properties>...</properties>

<!-- Build Settings -->

<build>...</build>

<reporting>...</reporting>

<!-- More Project Information -->

```
 <name>...</name>
 <description>...</description>
 <url>...</url>
 <inceptionYear>...</inceptionYear>
 <licenses>...</licenses>
 <organization>...</organization>
 <developers>...</developers>
 <contributors>...</contributors>
  <!-- Environment Settings -->
 <issueManagement>...</issueManagement>
 <ciManagement>...</ciManagement>
 <mailingLists>...</mailingLists>
 <scm>...</scm>
 <prerequisites>...</prerequisites>
 <repositories>...</repositories>
  <pluginRepositories>...</pluginRepositories>
 <distributionManagement>...</distributionManagement>
 <profiles>...</profiles>
</project>
```

The cornerstone of the POM is its dependency list. Maven downloads and links the dependencies for you on compilation and other goals that require them. As an added bonus, Maven brings in the dependencies of those dependencies (transitive dependencies), allowing your list to focus solely on the dependencies your project requires.

One powerful addition that Maven brings to build management is the concept of project inheritance. Although in build systems such as Ant, inheritance can certainly be simulated, Maven has gone the extra step in making project inheritance explicit to the project object model.

Similar to the inheritance of objects in object oriented programming, POMs that extend a parent POM inherit certain values from that parent.

Setup

Download it here: http://maven.apache.org/download.html

These are instructions for Windows, for linux/mac or more detail there are tutorial slides posted on moodle.

Three steps:

1) Add the environment variable M2_HOME and set it to the file path where you placed the unzipped Maven folder, ex: “C:\Program Files\Apache Software Foundation\apache-maven-3.0.4”

2) Add the environment variable M2 and set it to “%M2_HOME%\bin”

3) And then make sure you have the environment variable JAVA_HOME set to your JDK directory

NetBeans 7.0+ is packaged with a Maven plugin, the installation above is for use on the command line. I found it easier to add libraries to the Maven repository with the command line, but for everything else I just used NetBeans.

Here is an example of adding a library to the repository (from command line):

```
mvn install:install-file -DgroupId=com.mysql.jdbc -DartifactId=com.mysql.jdbc -Dversion=5.1.18 -Dpackaging=jar
-Dfile=C:\Users\redminion\Documents\School\Cmpt\JAVA\mysql-connector-java-5.1.18-bin
```

Report

Getting started was easy because NetBeans is packaged with a Maven plugin with is great, but I found it easier to manually all the libraries we wanted use from the command line. This was a bit tricky at one point because I'm running Windows so GlassFish is an installed program. There is a plugin that apparently works with the GlassFish install and Maven but I couldn't get it working, so I had to download GlassFish again but as a fully embedded jar which I could add to the Maven repository.

Haven't done anything with the build options because I don't know much about servlets and .JSP's but Maven builds and test (JUnit) our project nicely.

Maven is mostly meant for very large projects with many build options and a comprehensive test suite, so there isn't a lot we gain from using it for our project. Looking after dependencies is always a pain so it's great that Maven can do that for us, but even with GlassFish we only have 4 dependencies in total for our project.