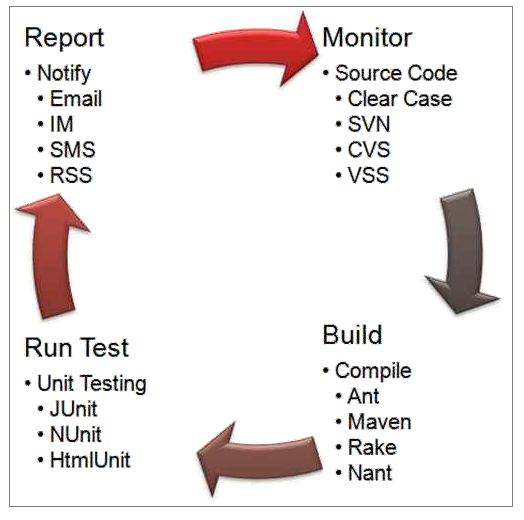
**Apache Maven:**

* s/w project mgmt. based on concept of POM (Project Object Model)
* manages a project’s build, reporting and documentation from a central piece of info
* provides developers with a complete build life cycle framework.



**Jenkins:** Open source continuous integration server. Has plugins to support building and testing virtually any project.

**Sonar:** quality measurement and reporting tool which produces different quality metrics (code coverage, deviations from standards etc.).

**Objectives:**

* Make build process easy
* Provides uniform build system
* Provides quality project info
* Reduces dev efforts time to make builds

All build systems are the same: Compile > Copy resource > Compile & Run Tests > Package > Deploy > Cleanup

Maven doesn’t have a concept of a condition (like if else).

**Build Tools:**

1. **Ant:** Multiple targets can be chained to combine single units of work into full workflow. First released in 2000.

**Disadvantages w.r.t. Maven:**

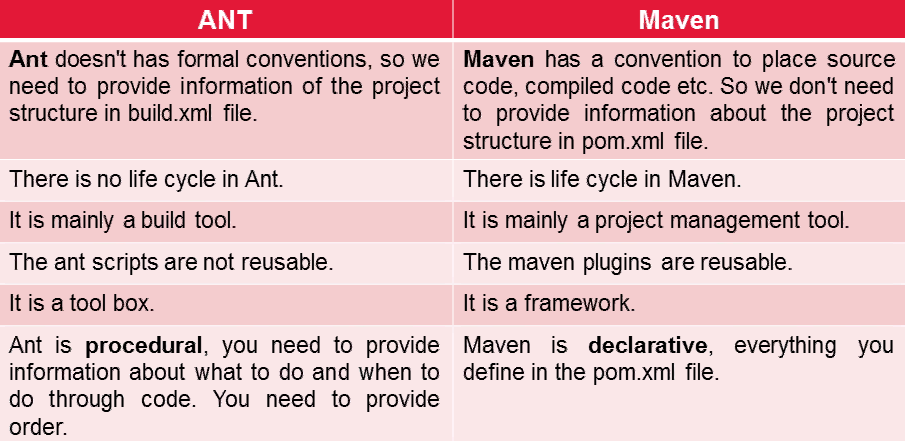
* Doesn’t give guidance on how to structure your project.
* Ext libraries - need to check into version control, no automated way of pulling them from central location.

1. **Maven:** Maven 2 released in 2005.

* Provided standardized project and directory structure + dependency mgmt.
* Project consists of multiple modules (can have dependency on each other).

1. **Gradle:** declarative, readable and clearly express their intention.

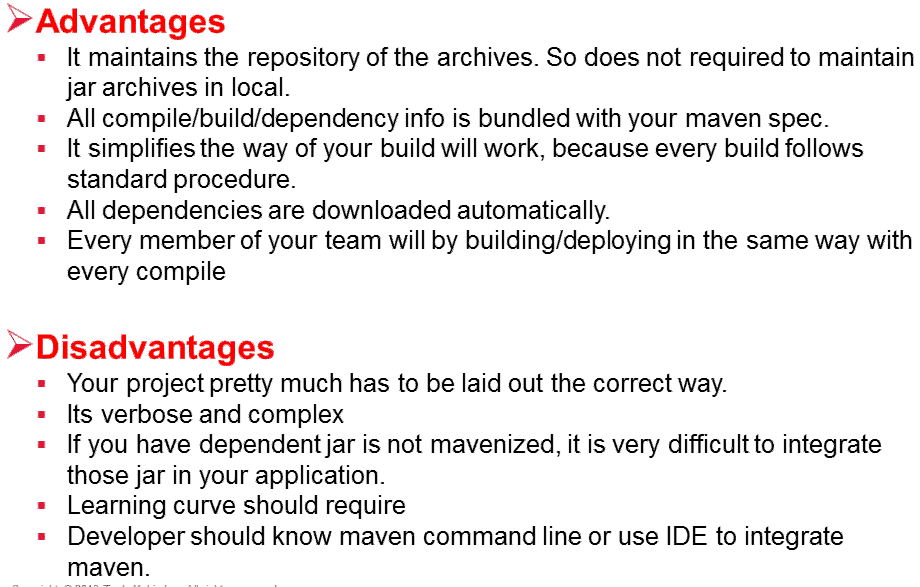
**Maven vs Ant:**



**Problems without Maven:**

* Add dependent jars (like sturts, spring, hibernate framework jars) in each project.
* Create right project structure

**Advantages and disadvantages:**



**Features:**

1. Dependency system
2. Multi-module builds
3. Consistent project structure
4. Consistent build model
5. Plugin oriented
6. Project generated sites
7. Profiling

**Maven POM:**

**Project Object Model** = XML based

Has:

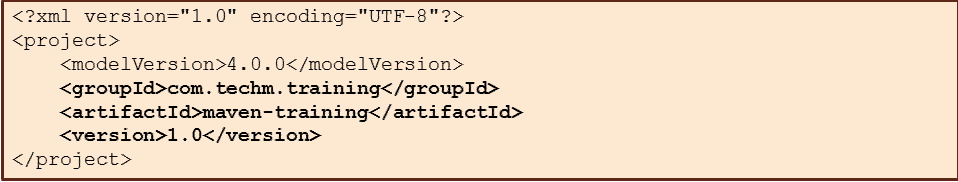
* Project Name and Version
* Artifact Type (like jar, war etc)
* Source Code Locations
* Dependencies
* Plugins
* Profiles (Alternate build configurations)

**Maven Project Name:**

GAV - uniquely identifies a project using

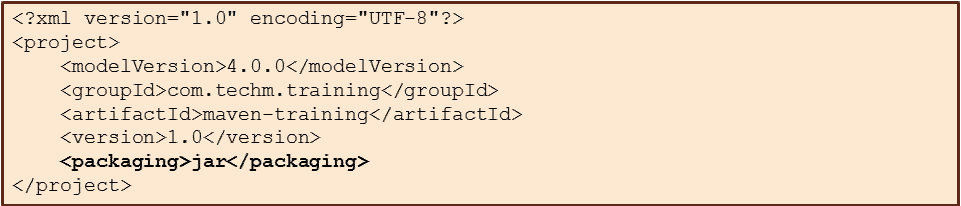
1. Group Id: project grouping identifier
2. Artifact Id: name of project (war file name, jar file name etc)
3. Version: Project version
   1. Format {Major}.{Minor}.{Maintenance}
   2. Add “-SNAPSHOT” to identify in development

Syntax - groupid:artifactid:version



**Element “Packaging”:**

* Specifies build type
* Tells how to build the project.
* Ex: pom, jar, war, ear, custom. Default is jar.



**Maven folder structure:**

* src
  + main
    - java: source files
    - webapp: web source files
    - resources: non-compiled source files
  + test: sources contributing to test a project.
    - java: test source files
    - resources: non-compiled test files
* target: default work dir

**Maven build cycle:**

* Default life cycle :
  + generate-sources/generate-resources
  + compile
  + test
  + package
  + integration-test (pre and post)
  + install
  + deploy
* There is also a **clean** life cycle.

**Maven Goals:**

Can set a lifecycle “goal”

* mvn install:
  + generate \*
  + compile
  + test
  + package
  + integration-test
  + install
* mvn clean
* mvn clean compile
  + cleam
  + generate \*
  + compile
* mvn test clean
  + generate \*
  + compile
  + test
  + clean

**Installation:**

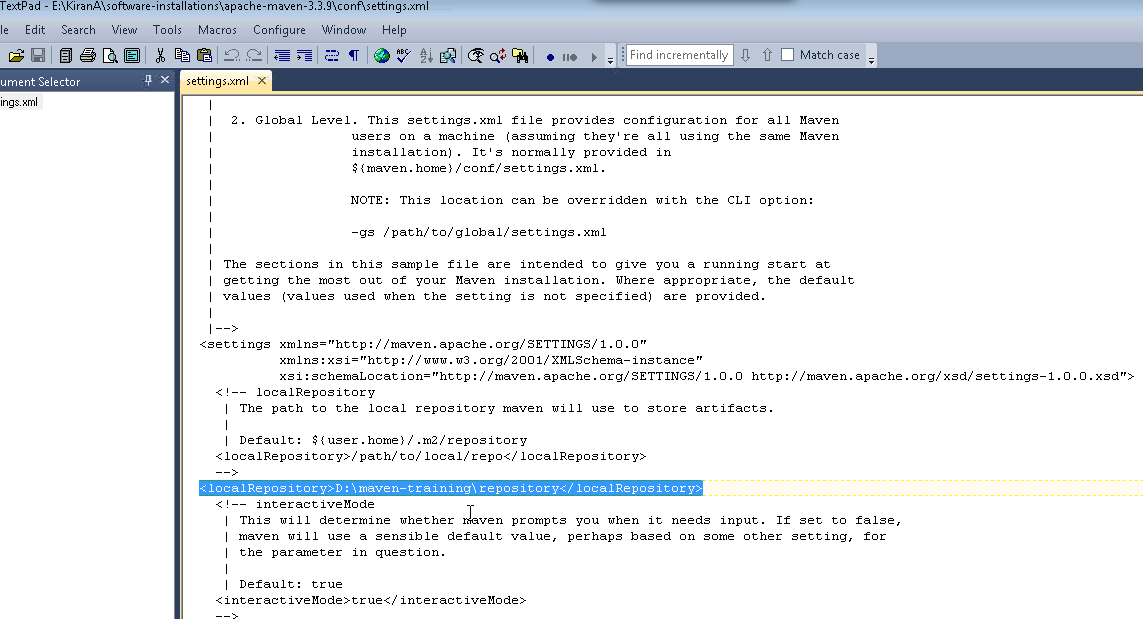
* Install maven from Apache site
* Add maven lib path in Path so that mvn command can work from cmd promt
* Install eclipse and install maven plugin

**Maven settings file:**

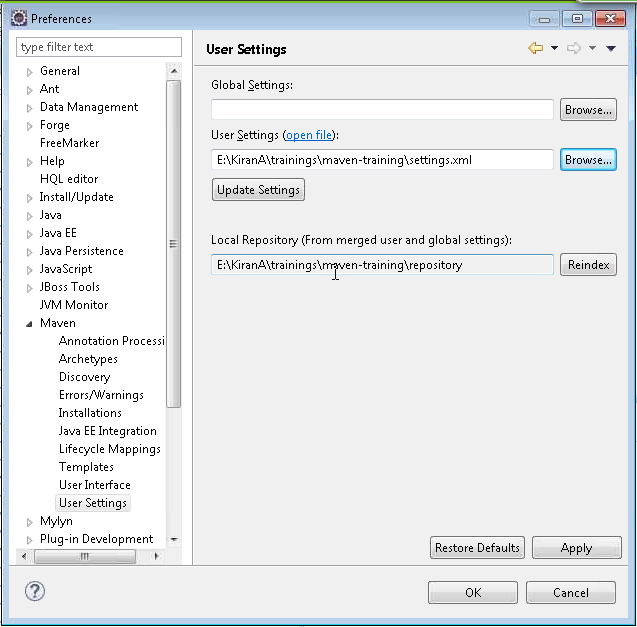
* Has elements for defining values/configurations for executing maven project in different ways.
* Should not be bundled with any specific project.



* Different settings can be controlled from mvn installed dir/conf/settings.xml file like default or user-defined local repository.

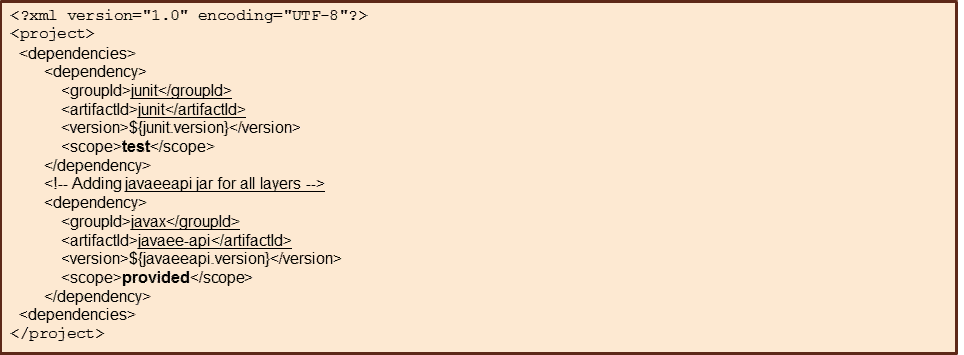


* Local repository keeps project’s all dependencies (lib jars, plugin jars etc.).
* Settings file that your eclipse project is using can be changed using Preferences.



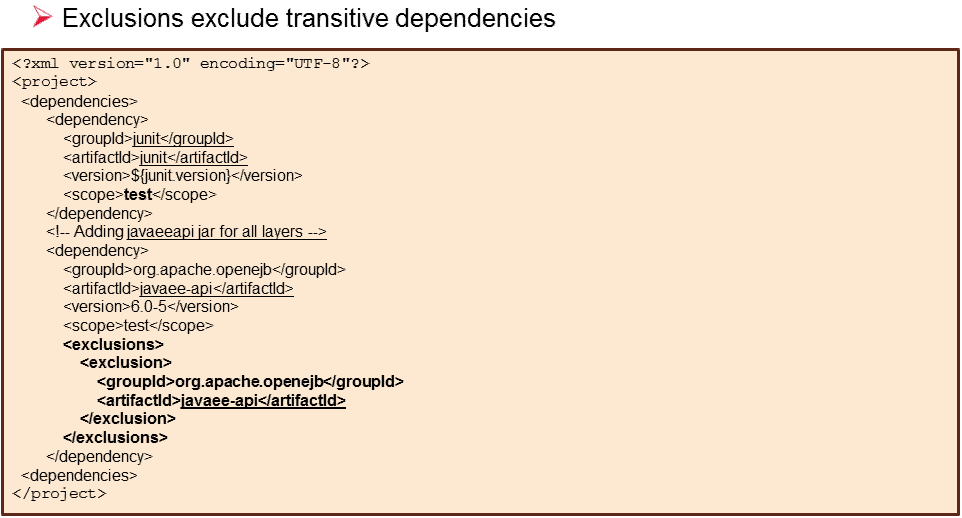
**Maven Dependency:**

* Maven introduced concept of **transitive dependency**.
* Dependencies consist of:
* GAV
* Scope: compile, test, provided (default=compile), runtime, system
* Type: jar, pom, war, ear, zip (default=jar)

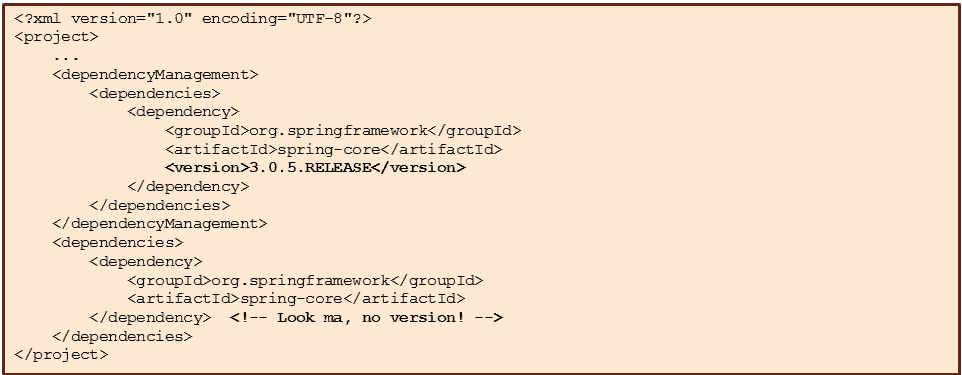


**Transitive dependency:**

* Project B > Project A; Project C > Project A => Project B is automatically included). Only **compile** & **runtime** scopes are transitive.
* Controlled using
  + Exclusions
  + Optional declarations



* **Dependency Management:** when more than one project depends on a specific dependency, it can be part of Dependency Management. Also takes care of version collision.



What is “provided” scope for a maven dependency?

The <scope> element can take 6 values: *compile*, *provided*, *runtime*, *test*, *system* and *import*.

This scope is used to limit the transitivity of a dependency, and also to affect the classpath used for various build tasks.

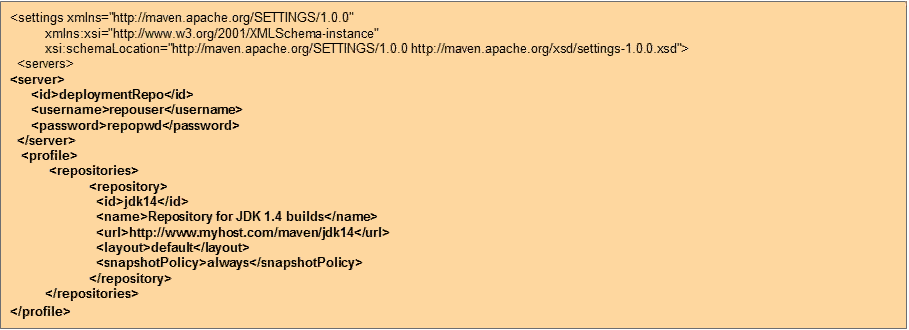
**Compile scope means that you need the JAR for compiling and running the app.** For a web application, as an example, the JAR will be placed in the WEB-INF/lib directory.

**Provided scope means that you need the JAR for compiling, but at run time there is already a JAR provided by the environment so you don't need it packaged with your app.** For a web app, this means that the JAR file will not be placed into the WEB-INF/lib directory.

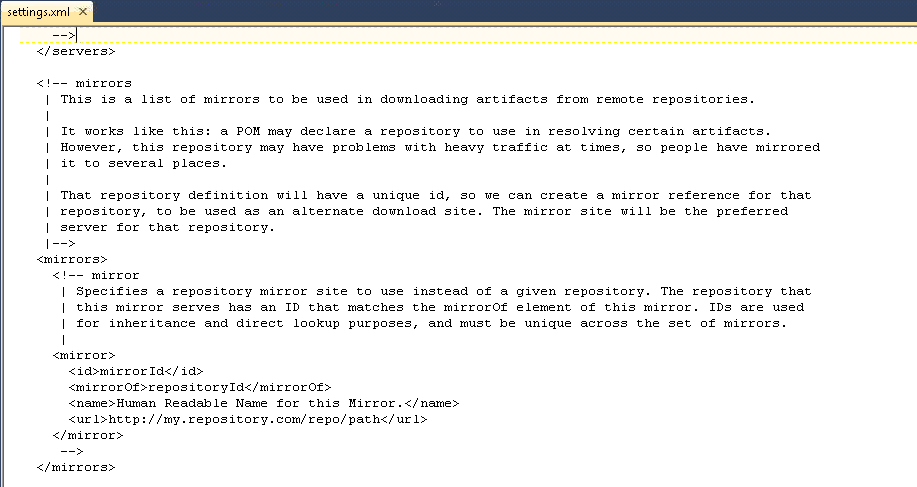
For a web app, if the app server already provides the JAR (or its functionality), then use "provided" otherwise use "compile".

**Maven repository:**

* Stores project jars, lib jars, plugins or any other project specific artifacts that can be used by Maven easily.
* Types:
  + Local
  + Central… ex.. machine accessible through internet
    - Provided by Maven community - with commonly used lib.
    - Can be configured in your local intranet system by using mirror config.



* + Remote… ex.. machine accessible through LAN
    - Repository accessed by variety of protocols - file:// and http://
    - Structured same way as local.. helps easy script run and offline sync.
    - Developer’s own custom repo - with required lib, other project jars.
* Maven searches for dependencies in the same order.
* Mirror config:

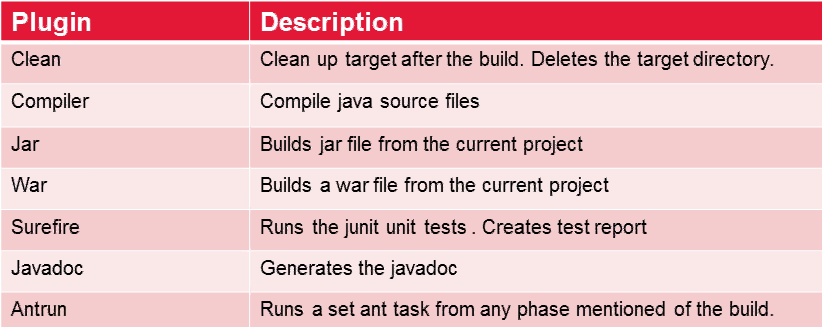


**Maven Plugins:**

* Maven => Plugin execution framework.
* Every task is done actually by plugins.
* Plugin => provides goal/s to be executed.
* Specified using plugin element in pom.xml
* Execution format:

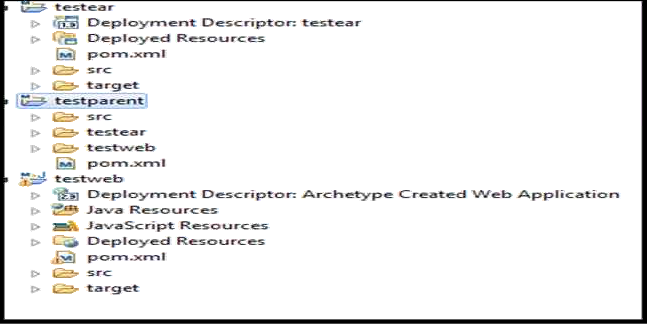
mvn <plugin name> : <goal name>

Ex. mvn compiler: compile



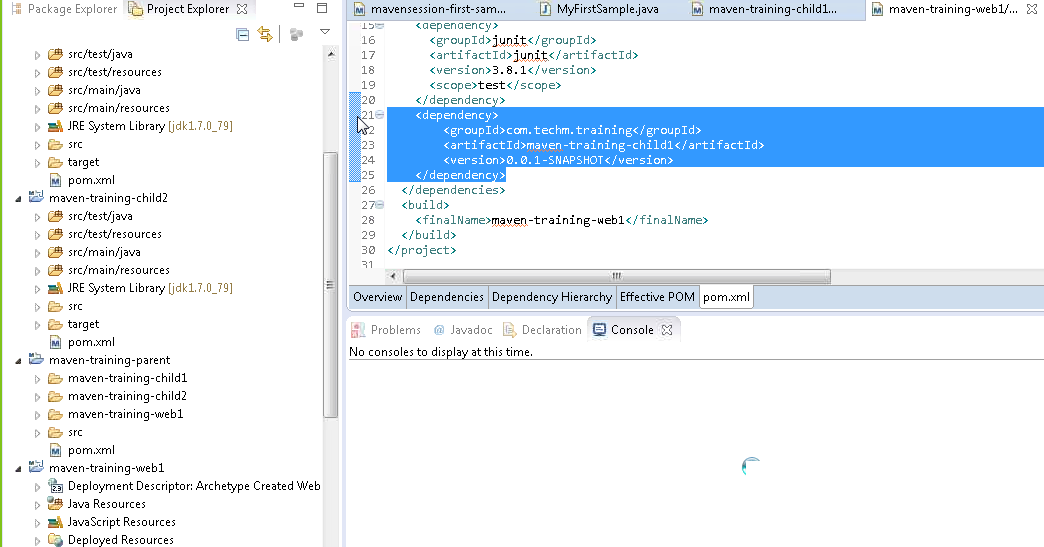
**Maven Project Structure:**

* Parent folder
* Ear folder
* Web folder

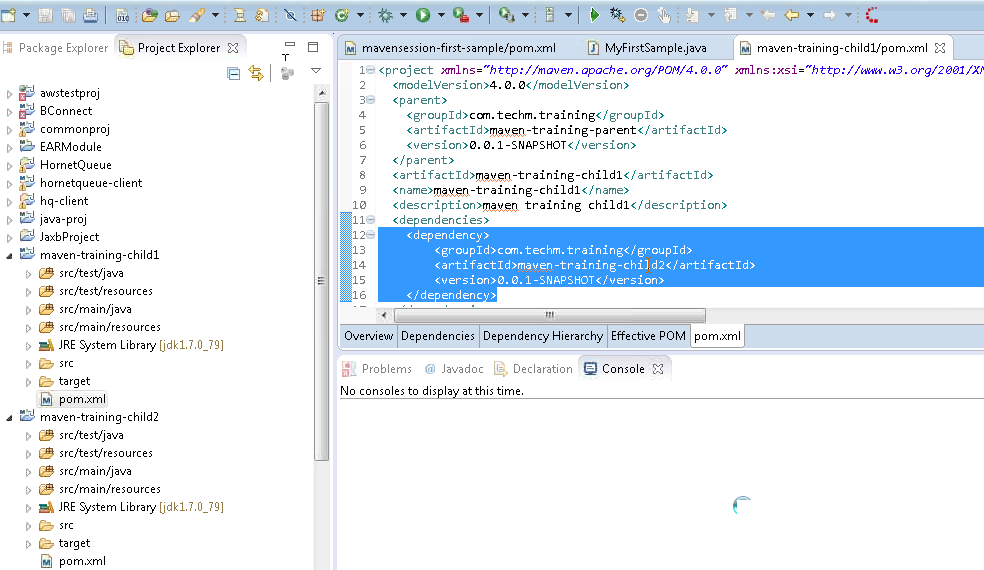


Example:

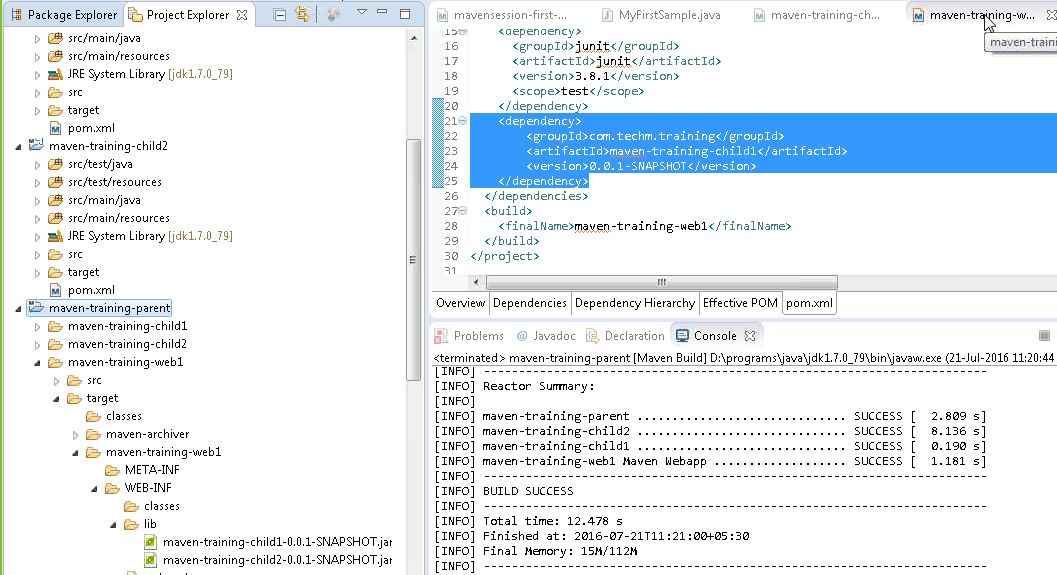
* Parent project (web1) has dependency on child1.



* Project child1 has dependency on child2 project.

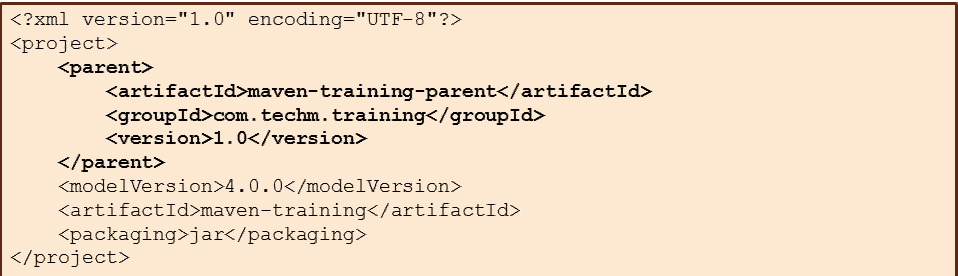


* child2 jar is automatically included.



**Project inheritance:**

* POM files can inherit:
  + group id, version
  + Project config
  + Dependencies
  + Plugin config



* Maven do support multi-module.
* Parent POM is used to group modules.



**Maven Build Profiles:**

* Set of config values for setting/overriding default values of maven build.
  + Can customize build for different env .. prod vs dev
  + Specified in POM file
  + Modify POM at build time
* Can be activated in different types:
  + Explicitly - using cmd console input.
  + Through maven settings.
  + Based on env variables (user/system var)
  + OS Settings (ex. WINDOWS family)



