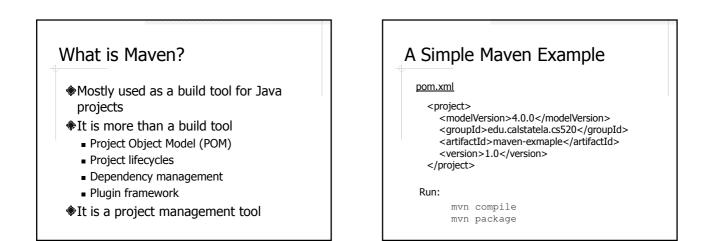
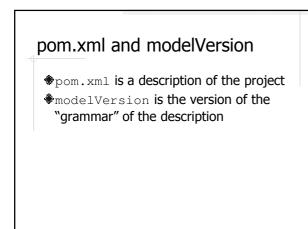
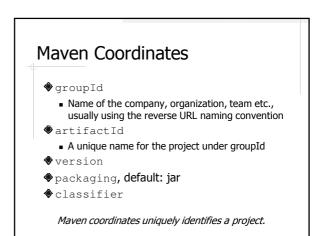


Build

- Preprocessing
- Compilation
- Postprocessing
- Distribution
- Deployment

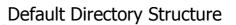








Systems, libraries, and frameworks should assume *reasonable defaults*.



- \$src/main/java
- \$src/main/resources for files that
 should be placed under classpath
- \$src/main/webapp for web
 applications
- src/test/java
- target

Build Lifecycle

- The process for building and distributing a project
- A build lifecycle consists of a number of steps called phases.

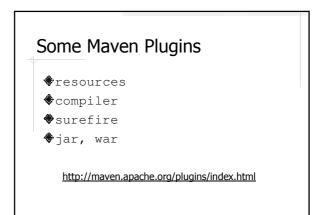
Some Default Lifecycle Phases

//maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#Lifecycle Reference

- validate
- compile
- ♦test
- package
- deploy

Goals and Plugins

Goals, a.k.a. Mojos, are operations provided by Maven plugins



Example of Using a Plugin	
 <build><plugins><plugin> <groupid>org.apache.maven.plugins</groupid> <artifactid>maven-compiler-plugin</artifactid> <version>2.3.2</version> <executions> <id>default-compile</id> <pre>cypals> <gpals> <gpals> <gpals> <configuration> </configuration> </gpals></gpals></gpals></pre></executions></plugin><!--</th--><th></th></plugins></build>	

About The Plugin Example

- A plugin is uniquely identified by its coordinates just like any other project
- Goals are usually associated (i.e. bound) to a build lifecycle phase
- The behavior of a goal can be customized with additional parameters in the <configuration> section

Run a Maven Build

mvn <phase>

- Maven will go through each build lifecycle phase up to the specified phase
- In each phase, execute the goals bound to that phase

Run a Maven Build in Eclipse

- Need the m2e Eclipse plugin
- ♦Right click on the project then select Run As → Maven Build ...
- Give the build a name
- Enter the phase name for Goals
- Click Run

Why Not Just Use an IDE

Can your IDE do everything you want?

- Deploy a web application to a remote server
- Generate source code from some metadata files
- Create a zip package of selected files for homework submission
- ...

Why Use Maven

- Everybody uses it!
- Common framework for project build and management
 - Project Object Model
 - Build lifecycles
- Archetype
- Dependency management
- Resource filtering

Archetype

- An archetype is a *template* for a Maven project which can be used to create new projects quickly
- Example: creating a project from archetype
 - maven-archetype-quickstart
 - maven-archetype-webapp
- Users can create new archetypes and publish them through catalogs
 - Main Maven archetype catalog: http://repo.maven.apache.org/maven2/archetype
 - catalog.xml

Dependency Management

- A dependency of a project is a library that the project depends on
- Adding a dependency to a project is as simple as adding the coordinates of the library to pom.xml
- Maven automatically downloads the library from an online repository and store it locally for future use

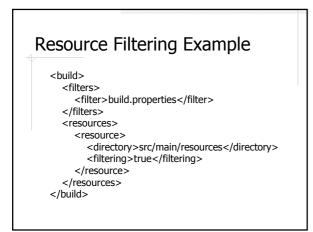
Dependencies and Repositories

- Search for dependency coordinates at <u>http://mvnrepository.com/</u>
- Maven Central Repository -<u>http://repo.maven.apache.org/maven2/</u>
- Additional libraries and repositories -<u>https://maven.nuxeo.org/</u>

More About Dependency Management

- Dependencies of a dependency are automatically included
- Dependency conflicts are automatically resolved
- See CSNS2 for example

Resource Filtering ©Use placeholders in resource files and replace them with actual value during the build process <param name="File" value="\${app.dir.log}/csns2.log" /> param name="File" value="F:/TEMP/csns2/csns2.log" />



Summary

- Project Object Model (POM)
- Coordinates
- Lifecycles and phases
- Plugins and goals
- Archetype
- Dependency management
- Resource filtering

Further Readings

Maven: The Definitive Guide by Sonatype