Maven Project Management

NetDB

CS, NTHU,

Fall, 2013

Agenda

- Into the wild through Apache Maven
  - Directory and archetypes
  - Build lifecycles
  - pom.xml
- Your first open source Integration
  - Apache HttpClient

Why Maven?

- **Maven** is a build automation tool used primarily for Java projects
- Core concepts
  - Convention over configuration
  - Dependency management
  - Plugin-based
  - Project Object Model

Get Maven

- The Maven framework is build-in in the latest eclipse Kepler.
- If the eclipse version in your computer is not Kepler, please download it here

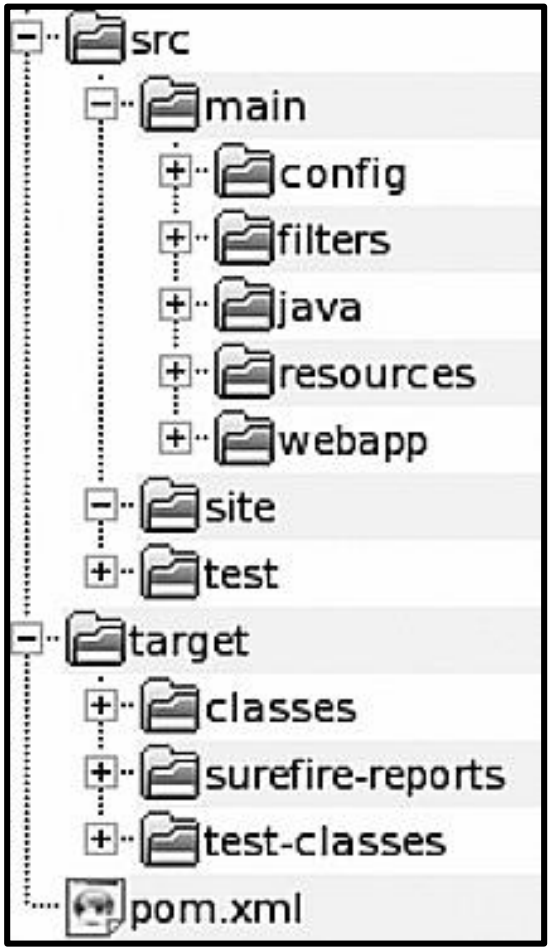
Creating a Maven Project

- In eclipse, go to File -> New -> Other -> Maven -> Maven Project
- Keep clicking “Next” until it ask you to type the Group ID and Artifact ID

Creating a Maven Project

- Archetype:
  - A project layout template (remember src/main, src/test, target/classes etc.?)
  - “maven-archetype-quickstart”
- Group ID:
  - Unique among organizations and projects (like Java package)
  - “netdb.courses.softwarestudio” (or “com.microsoft.windows”, etc.)
- Artifact ID:
  - Your project name
  - “http-retrieve” (or “winxp”, “win7”, etc.)
- Version:
  - Format: <major>.<minor>.<revision>([-<qualifier>]|[-<build>])
  - “0.0.1-SNAPSHOT” (or 7.0.0, 7.1.1-beta-2589436513, etc.)
- Packaging:
  - What your project end up with
  - “jar” (or war, ear, etc.)

Maven Directory Layout

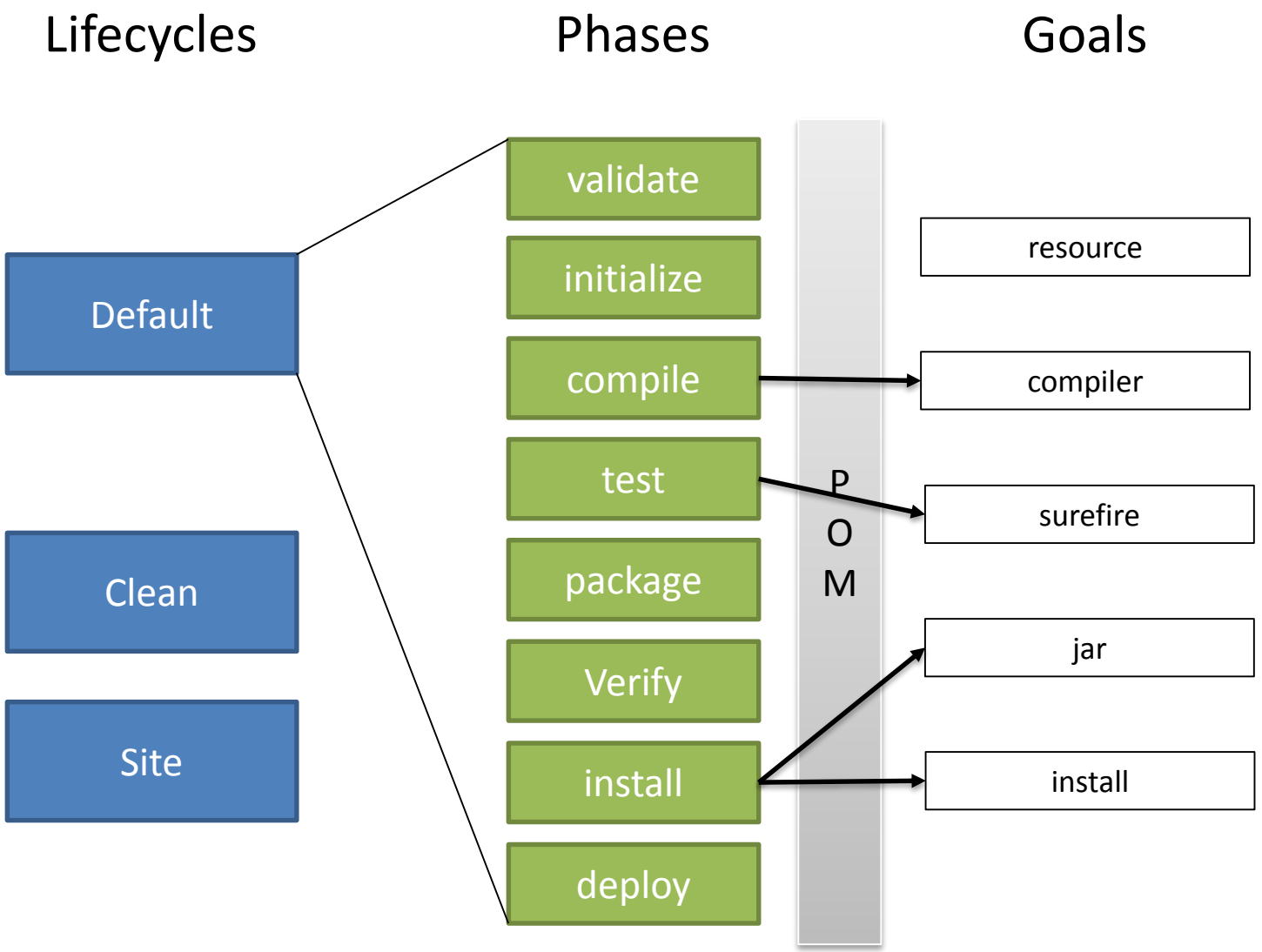


- src/main/java
  - Where your *.java files go
- src/main/resources
  - Resources (e.g., language files) your codes need
- src/main/filters
  - Resource filters, in the form of properties files
- src/main/config
  - Configuration files (properties or XML)
- src/main/webapp
  - Web directory containing XHTML, CSS, and javascript files
- src/test
  - Has same structure as src/main but contains files for testing purpose only
- src/site
  - Files used to generate maven documentation/reports
- target/*
  - Where the output files go

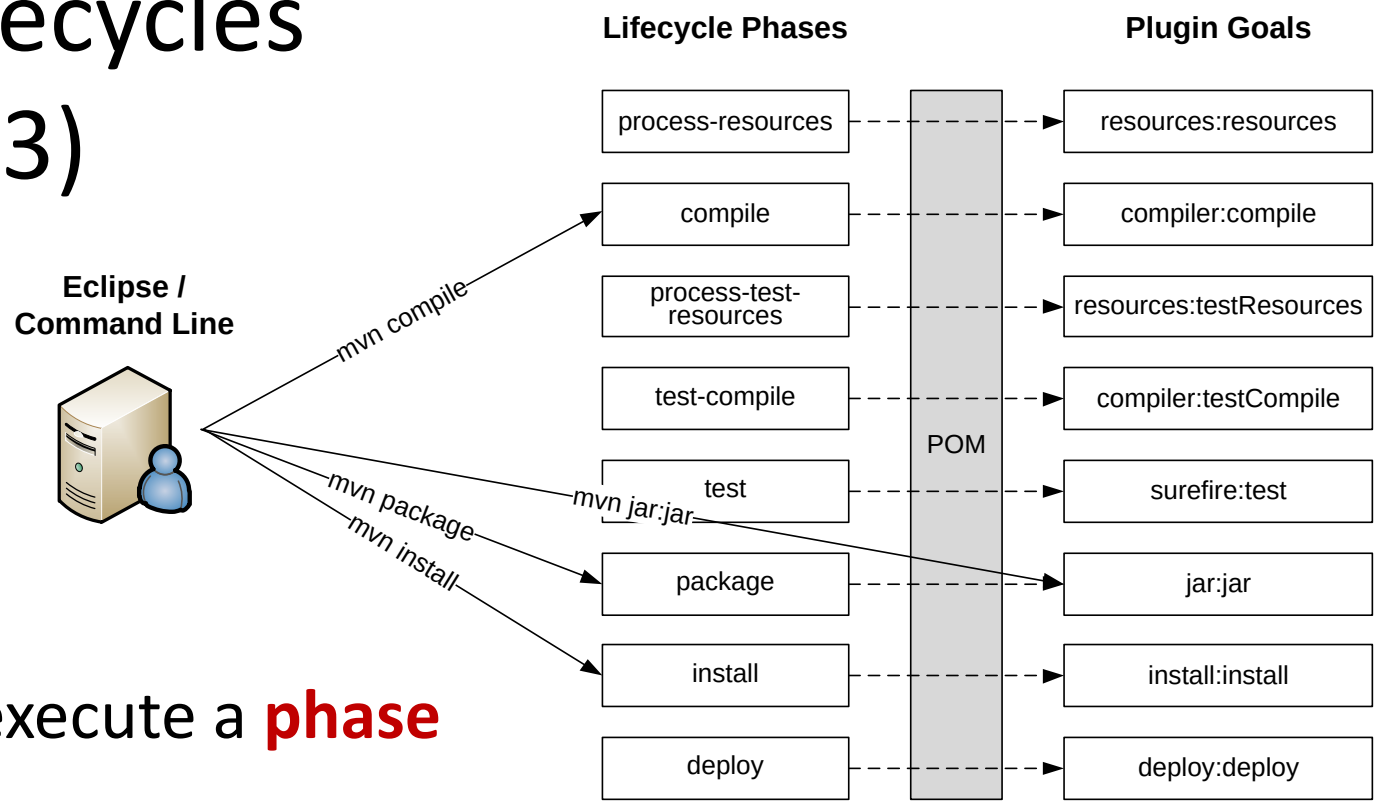
Archetypes

- It is tiresome to create a full set of empty directories when starting a new project
- Maven knows
  - Provides the archetype plugin
  - Creates a Maven compatible layout quickly based on your project needs
- Common archetypes:
  - maven-archetype-quickstart, maven-archetype-simple (for Java applications with `main()`)
  - maven-archetype-webapp, or gae-archetype-objectify-jsp (for web applications)
- See all Built-in Archetypes

Build Lifecycles (1/3)



Build Lifecycles (2/3)



- You can execute a **phase** or a **goal**
  - Executing a phase will result in execution of all preceding phases
- Each phase may be associated with zero or more goals
  - Association is specified in pom.xml
- **Plugins** are special maven projects that provides goals

Build Lifecycles (3/3)

- Three built-in lifecycles: default, clean, and site
  - See documentation
- Most frequent command: `mvn clean install`
  - The clean phase (and precedents) of the clean lifecycle is invoked first
  - Then the install phase (and precedents) of the default lifecycle
- Let's play with the project
  - Project is "installed" to local repository
- For prospective gurus:
  - You can define your own archetypes
  - You can develop your own plugins and goals (in "mojo" projects)

Installation

- How is my project installed?
  - The packaged jar was moved from ${proj_path}/target/ to your local repository (at ${user_root}/.m2/repository)
- Path:

  - Common classifier: none, `sources`, or `javadoc`
- So what?
  - This project, as other maven projects, is ready to be reused (locally)
  - You can make a new project “depend on” this project

The POM (1/3)

- Project Object Model (POM) is the heart of Maven
  - Specifies the association between lifecycle phases and plugin goals
  - Specifies how a project should depend on other projects
- You need to know how to write **pom.xml**
  - You all have added dependencies for your project before!

The POM (2/3)

- Take a look at the pom.xml of the “geomap” project

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

4.       <groupId>netdb.courses.softwarestudio</groupId>
5.       <artifactId>geomap</artifactId>
6.       <version>1.0-SNAPSHOT</version>
7.       <packaging>jar</packaging>

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

11.      <dependencies>
12.              <dependency>
13.                      <groupId>junit</groupId>
14.                      <artifactId>junit</artifactId>
15.                      <version>4.9</version>
16.                      <scope>test</scope>
17.              </dependency>
18.      </dependencies>
```

The POM (3/3)

```
19.    <build>
20.       <plugins>
21.          <plugin>
22.             <groupId>org.apache.maven.plugins</groupId>
23.             <artifactId>maven-compiler-plugin</artifactId>
24.             <version>2.3.2</version>
25.             <configuration>
26.                 <source>1.6</source>
27.                 <target>1.6</target>
28.              </configuration>
29.           </plugin>
30.           <plugin>
31.              <groupId>org.apache.maven.plugins</groupId>
32.              <artifactId>maven-surefire-plugin</artifactId>
33.              <version>2.9</version>
34.              <configuration>
35.                 <systemProperties>
36.                     <property>
37.                         <name>java.util.logging.config.file</name>
38.                         <value>${project.build.directory}/test-
   classes/java/util/logging/logging.properties</value>
39.                     </property>
40.                 </systemProperties>
41.              </configuration>
42.            </plugin>
43.         </plugins>
44.     </build>
45.</project>
```

A Closer Look

- What's inside pom.xml?
  - Basic info (e.g., ids, version) of your project
  - Packaging (e.g., jar, war)
  - Build process (e.g., lifecycle phases, goals, plugins)
  - Dependencies (e.g., reusable maven projects including yours)
  - Repositories, Reporting, etc.
- The `modelVersion` element specifies POM version and is always required

Project Description Elements

- `groupId`:
  - Unique among organizations
  - Acts like Java package structure in repository
- `artifactId`:
  - Name of the project
  - Along with groupId, it create a unique path for a project in the world
- `version`:
  - Format: <major>.<minor>.<revision>([-<qualifier>]|[-<build>])
  - E.g., 0.0.1-SNAPSHOT, 1.0.0, or 2.1.1-2589413
- `packaging`:
  - Defines the default mapping between lifecycle phases and plugin goals
  - Default to jar, but can be others (e.g., war)

Properties

- Each child element of properties defines a property
  - Used by ${prop_name} elsewhere in POM
- Default properties:

Property	Description
`project.*`	Reference any value in POM
`settings.*`	Reference values in ${user_root}/.m2/settings.xml
`env.*`	Environment variables like PATH and JAVA_HOME
`java.*, os.*`	Properties from java.lang.System.getPropertys()
`basedir`	Directory containing pom.xml

Build Element

- Controls how each lifecycle phase is mapped to plugin goals (in addition to packaging)

```
1. <properties>
2.     <jave.source.compatibility>1.6</jave.source.compatibility>
3.   </properties>
4. ...
5. <build>
6. ...
7.   <plugins>
8.     ...
9.     <plugin>
10.      <groupId>org.apache.maven.plugins</groupId>
11.      <artifactId>maven-compiler-plugin</artifactId>
12.      <configuration>
13.        <source>${java.source.compatibility}</source>
14.        <target>${java.source.compatibility}</target>
15.      </configuration>
16.    </plugin>
17.  </plugins>
18.</build>
```

Plugins

- How can I package my source codes into a jar with classifier `sources`?
  - The jar plugin doesn't do that, you need another
- The "source" plugin (see maven-source-plugin) packages the source code
  - Goal: source:jar
- But we have to manually bind this goal to our lifecycle

```
<build>
  ...
  <plugins>
    ...
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-source-plugin</artifactId>
      <executions>
        <execution>
          <id>attach-sources</id>
          <goals>
            <!-- binds to the package phase by default -->
            <goal>jar</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

Dependencies

```
1. <dependencies>
2.   ...
3.   <dependency>
4.     <groupId>org.apache.httpcomponents</groupId>
5.     <artifactId>httpclient</artifactId>
6.     <version>${httpclient.version}</version>
7.     <type>jar</type>
8.     <classifier>sources</classifier>
9.     <scope>provided</scope>
10.  </dependency>
11.  <dependency>
12.    <groupId>junit</groupId>
13.    <artifactId>junit</artifactId>
14.    <version>${junit.version}</version>
15.    <type>jar</type>
16.    <scope>test</scope>
17.  </dependency>
18.</dependencies>
```

- Common `classifier`: none, `sources`, or `javadoc`
- `type`: Corresponding to dependent artifact's packaging
  - Default to jar
- `scope`: compile (default) | provided | runtime | test | system
- Reuse your project by specifying dependency here
- Right click and click "Download Sources" to download all the sources of dependencies (if provided)

Misc.

- Profiles Element
- Reporting Element
- Repositories Element
- See POM Reference

Agenda

- Into the wild through Apache Maven
  - Directory and archetypes
  - Build lifecycles
  - pom.xml
- Your first open source Integration
  - Apache HttpClient

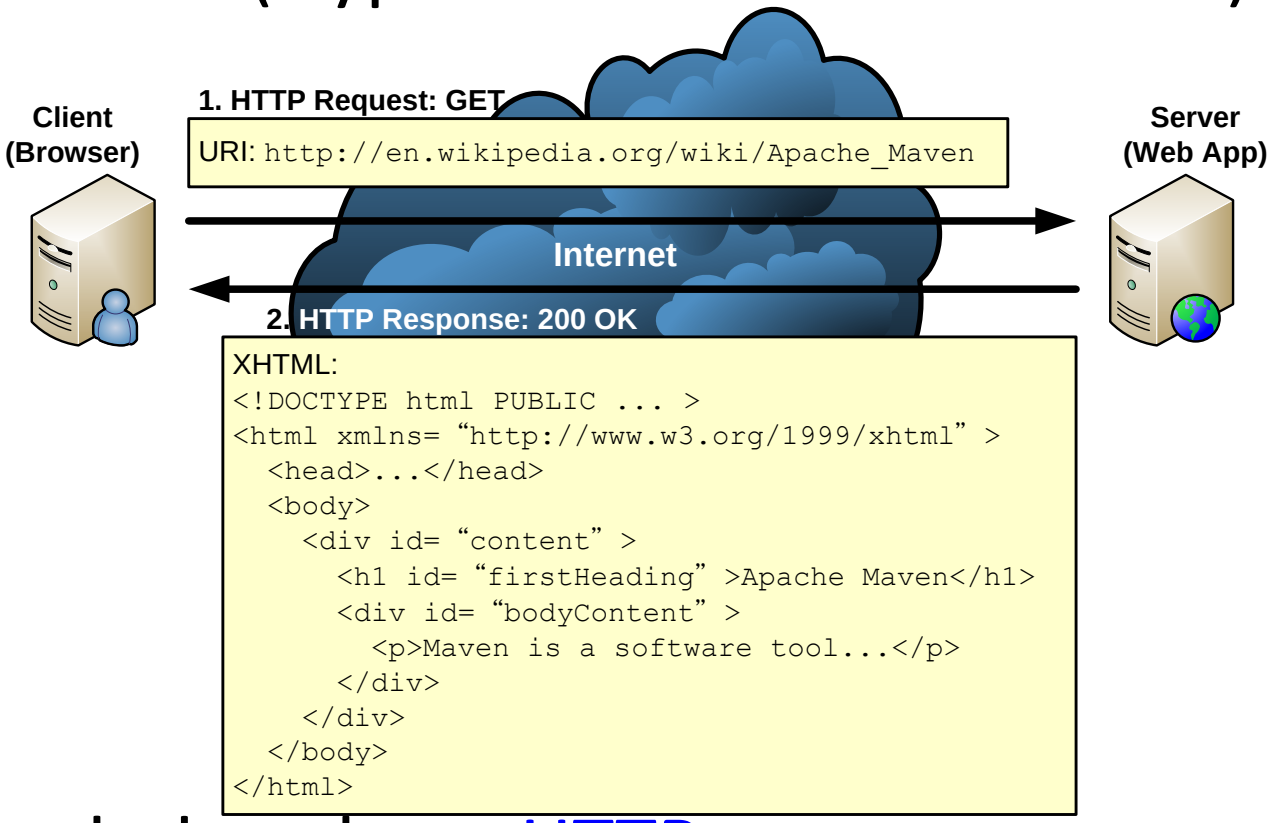
Open Source Integration

- What if I want to share my project with remote users?
  - Local repository is proprietary
- Maven hosts a central repository
  - You can upload your project to central repository
  - Path:

- It's easier to see what others have done
  - Browsing Maven Central Repository
  - Follow the artifact path

Apache HttpClient

- How does HTTP (Hypertext Transfer Protocol) work?



- Some knowledge about HTTP

Exercise

- Implement HttpRetrieve using Apache HttpClient to print out the XHTML contents of a given URL (e.g. http://en.wikipedia.org/wiki/Apache_Maven)
- Create a Maven project (archetype: “maven-archetype-quickstart”)
- Editing the POM.xml to add an dependency:
  - groupId: org.apache.httpcomponents
  - artifactId: httpclient
  - version: 4.3

App.class

- You should implement a HttpRetrieve class which receives a string as URL and prints out the html contents

```
1. public class App{
2.
3.     public static void main(String[] args) {
4.
5.          Httpretrieve httpr = new Httpretrieve(args[0]);
6.          httpr.retrieveHttp();
7.
8.     }
9. }
```

Hint

- See this Apache HttpClient Tutorial on how to open an InputStream to read out the content
  - Section 1.1 should be enough for current task
- Read Java API on `java.io.*` how to use InputStream

```
1. InputStream input = ... // you will get an InputStream instance from
   HTTP Client
2. int data = input.read();
3. while(data != -1) {
4.     ... // data contains a char. Cast it to char and print it out
5.     data = input.read();
6. }
7. input.close();
```