Jar

Packaging .class files into single file

What is a Jar File?

- Java archive (Jar) files are compressed files that can store one or several files
  - It is actually a `.zip` file, just named differently
  - For example: `myclasses.jar`

- Jar files normally contain Java `.class` files
  - Other files may also be included, like images, java source files etc…

- Jar files are runnable on Windows if JRE is installed
  - Must contain a class with a `main()` method
  - E.g. a Swing app can be packaged in a `.jar` file and executed by double-clicking the file

JAR File – Why?

- Jar files are compressed
  - Reduce the size of the original files
  - Related files are grouped together in a logical group

- For example an applet can be downloaded in one HTTP request instead of several
  - Faster!

- Jar files can be signed digitally
  - Users who recognize the signature can optionally grant permission to run the code or refuse it
  - Many kinds of security rules / restrictions can be applied

How to Create a Jar File

- The `jar` command utility comes with the JDK
  - Can be found in the JDK `bin` folder
  - You could also let your favorite IDE create the Jar file for you…

- The basic format to create a Jar file is:

```
jar cf jar-file input-files
```

- Assume we have an application with two classes
  - `FooSwing.class`
  - `Foobar.class`

- To make a Jar file called `Foo.jar` use the command

```
jar cvf Foo.jar FooSwing.class
Foobar.class
```

Execute a Jar Application

- If the Jar file contains a class with a `main()` method, it is a so called "executable Jar file"
  - Or simply an application

- Jar applications can be run with the following command:
  - `java -jar jar-file`

- To run the `Foo.jar` (in case it is an application)
  - `java -jar Foo.jar`

Manifest file

- The manifest is a special file that can contain information about the files packaged in a Jar file
  - By tailoring this "meta" information that the manifest contains, you enable the Jar file to be used for a variety of purposes
  - When you create a JAR file, it automatically receives a default manifest file
  - There can be only one manifest file in an archive, and it always has the path/name `META-INF/MANIFEST.MF`

- To customize the manifest, you must have a text file containing the information you wish to add
  - You then use the Jar tool's `m` option to add the information in your file to the manifest
  - The basic command to do this is:

```
jar cvmf myjar.jar manifest.mf A.class B.class
```

JavaDoc Tool

Documenation with comments

JavaDoc

- JavaDoc is a tool that generates Java code documentation from code comments
  - Can be run from command line or directly from an IDE (e.g. Netbeans, Eclipse)
- Input: Java source files (.java)
- Output: HTML files documenting specification of java code

Code File Documentation

- Documentation is defined in comment lines

```
/**
* This is an example of a
* simple 3 line documentation
* in a javadoc.
*/
```

- Comments are placed immediately before class, interface, constructor, method, or field declarations
  - Nothing between them is permitted
  - Comments are written in HTML

```
/**
* Just a <b>simple</b> comment.
* @see java.lang.Object
*/
```

Package-Level Comments

- Create a file called `package-info.java`
  - Place the file in the root folder of your Java code package
  - Separate file needed for each package – file name is always the same!
  - Before Java 1.5 you would need `package.html` instead

- Otherwise documentation is pretty similar to any other JavaDoc documentation
  - See example on the notes page

Class and Interface Comments

- Comments are placed immediately before class or interface declaration
  - Starts with /** and finishes with */
  - Comments are written in HTML
  - Comments are composed of main description
  - First sentence is a summary

- Tag section
  - @author - name and email
  - @version - version and date
  - @exception or @throws - exceptions of the class (optional)
  - @see - references (optional)

Class and Interface Comments

```
/**
* Summary of class/interface.
*
* <p>
* Main description of class/interface
* as HTML.
* </p>
*
* @author John Smith, fohn.smith@foo.com
* @version 1.4, 08/10/2008
* @see MyInterface
*/
public class MyClass implements MyInterface {
   // ...
}
```

Method and Constructor Comments

- Same as for class/interface comments

- Tags
  - @param - name of argument and description
  - @return - description (only for methods which return value is not void)
  - @throws- exception name and description
  - @see - reference (optional)

Method and Constructor Comments

```
/**
* Description of the method as HTML text.
*
* @param name1       description of name1
* @param name2       description of name2
* @return description of return value
* @throws Exception1 why it occurs
* @see     ReturnType
*/
public ReturnType myMethod(Type1 name1,
          Type2 name2) throws Exception1 {
   // ...
}
```

Field (Attribute) Comments

- Contains main description and tag @see

Example:

```
/**
* Something about the attribute.
*
* @see getMyMethod()
*/
int y = 2222;
```

Maven

Building for flexibility

What is Maven?

- Maven is a software project management and comprehension tool
  - It's built on top of ANT and uses a Project Object Model (POM)
  - Maven manages projects build, reporting and documentation using automated tasks
- Promote standardization, automation, and shared best practises and conventions

What can it do for you?

- It will help you manage
  - Builds
  - Documentation
  - Reporting
  - Dependencies
  - Source Control Management systems (SCMs)
  - Releases
  - Distribution

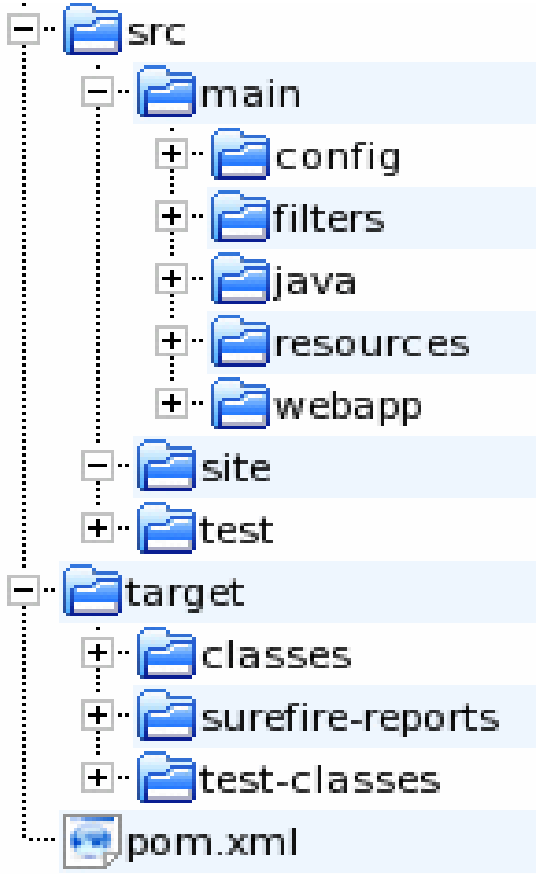
So how does it make you more efficient?

- To get started you create archetype project that typically contains
  - Precreated standardized folder structure
  - With possibility to build, package, document, and test your module with single command
  - With possibility to add things like IDE support and documentation site with single command
  - With ability to handle your **dependencies** so you don't need to put your compiled .jars in SCMS

Maven project structure

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- mycompany
    |               `-- app
    |                   `-- App.java
    `-- test
        `-- java
            `-- com
                `-- mycompany
                    `-- app
                        `-- AppTest.java
```

All standard folders
For Maven 2



Project structure

- Maven creates a pom.xml file in your project root folder
  - This is Project Object Model, and it's the main configuration file for your Maven setup
- Maven creates src/main/java folder for your java sourcecode
  - And src/test/java for your unit tests

Pom.xml file

```
<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.mycompany.app</groupId>

  <artifactId>my-app</artifactId>

  <packaging>jar</packaging>

  <version>1.0-SNAPSHOT</version>

  <name>Maven Quick Start Archetype</name>

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

  <dependencies>

    <!- references to extra libraries →

  </dependencies>

</project>
```

Phase and Goal

- Maven has two levels of commands.
- Phases are high-level commands, or life-cycle phases
- Goals are smaller goals inside a phase – phase may have several goals
- Phases have default goals that will be run if goal is not specified