Contents

• Maven
  What is maven
  Terminology
  Demo
• Log4j and slf4j
  What is logging
  Advantages
  Architecture
Maven

• What is maven?
• How does it work?
• A simple pom.xml
• Command line demo
• Creating eclipse projects (eclipse plugin for maven)
• Using maven within eclipse
What is maven?

maven.apache.org says:
Maven is a project management and comprehension tool and that helps with managing:
Builds (similar to Ant, but not identical)
Documentation
Reporting
Dependencies
Software Configuration Management (SCMs) git
Releases
Distribution
We only use dependencies and build.
Architecture of maven (simplified)

• Application
• Local Repository (for example: .m2/repository/)
• Maven Repository

Application pulls jar files from Local Repository and (central) Maven Repository
Copies from Maven Repository to local
You can also install to Local Repository and Maven Repository

Picture
Installatin

• Straight forward (see maven page)
• Shipped with Eclipse or can be added
Maven terminology

• Running archetype-generate produces the following
• Archetype: is a model on how you want your project to be structured.
• Version: is different versions of the archetype
• groupId: similar to packagename
• artifactId: similar to class name (name of the jar or war)
• package: what do you want it to be packaged to. For example jar
Maven Life Cycle (not all)

• compile: usual compile file
• test: all test cases are run
• Package: produces jar or war or...
• install: install the package to local repository (NOT installing into tomcat or...).
• deploy: You deploy to Maven Remote Repository.
[NOT deploy into application sever; confusing].
• clean: to get rid of whatever was created since generation(target ...)
Apache Log4j

• This will be a brief lecture: you can learn more from various tutorials and available bookd:
  • http://jakarta.apache.org/log4j
  • http://supportweb.cs.bham.ac.uk/documentation/tutorials/docsystem/build/tutorials/log4j/log4j.html
  • Samudra Gupta: Pro Apache log4j
What is logging

• Gupta’s book:
  • Logging is a **systematic** and **controlled** way of representing the **state** of an application in a **human-readable** fashion.

• State: value of some/all variables
• Most people use `system.out.println` or debuggers.
• But logs are different from both of the above.
What are the advantages of logging?

• Problem diagnosis
• Quick debugging
• Easy maintenance
• History
• Cost and time saving

I’d like to add the following from my research on **process mining**. what is PM? Load balancing, identifying bottlenecks, common scenarios.
Disadvantages

• Extra runtime overhead
• Programming overhead- more code can mean more errors
• Bad logs confuse people
• Imposes extra requirements (means allocation of resources= money)
logging is much better than println

1. You can turn a log off to focus on a special aspect of the system
2. `println` can not separate log-messages of different degree of importance
3. `println` just writes to console
Three main objects in log4j

• Logger: captures logging information.
• Appender: publishes logging information to various destinations (console, file, socket, ..)
  – For example, a **ConsoleAppender** object for printing logging information to a console.

Layout: The Layout object is used to format logging information in different styles (human-readable part!)
Interaction of the three objects:
How does it work in practice?

• Three ways to **configure** log4j
• **Properties file**- log4j.properties
• XML style of representation
• Programmatic style

We choose the first one- for the other two methods of configuration see http://jakarta.apache.org/log4j
How does it work in practice?

• 1. Name a Logger object for the class
  ```java
  private static final Log LOG = LogFactory.getLog(Main.class);
  ```

• 2. Insert desired txt with level logging via Level-Based Logging Methods
  ```java
  LOG.fatal("House is on fire");
  ```

• 3. Specify Level of logging for the directory and type of Appender in log4j.properties
How does it work in practice?

• Example:
  
  `log4j.logger.org.hibernate.SQL=debug,stdout`

  Logging related to `org.hibernate.SQL` will be carried out at the level of "debug" and will be appended to `stdout`.

• How does it work?
• What happens to subpackages?
• What if `org.hibernate` has different level?

We will answer these next.
Logger object

• The main object that you deal with is a Logger object- everything else behind the scene.

• A convenient way is to get it is:
  ```java
  private static final Log LOG = LogFactory.getLog(ClassNAME.class);
  ```

• Notice: we can assign the logger in properties file and then retrieve it using
  ```java
  public static LoggergetRootLogger();
  ```
  ```java
  public static LoggergetLogger(String name);
  ```
Level-Based Logging Methods

• `public void debug(Object message);`
• This method prints messages with the level `Level.DEBUG`.
• There are similar methods for other 7 Levels:
  • `ALL<DEBUG<INFO<WARN<ERROR<FATAL<OFF`
  • `ALL`: means all information will be logged
  • `OFF`: means nothing will be logged
• Each Level prints messages about higher level
• Explain!
Level-Based Logging Methods

• The configuration can be altered from the log4j.properties; So we can have different level of logging at different time. For example looking at only FATAL messages.

• There are also **Filter** objects to analyze logging information. **Filter** are beyond the level of this course, see log4j manual
Logger Hierarchy

• There is a hierarchy in 
  • `log4j.logger.org.hibernate.tool.hbm2ddl=debug, stdout` 
• as follows: 
  • `org.hibernate.`
  • `org.hibernate.tool`
  • `org.hibernate.tool.hbm2ddl`
• What if Level for some layers are not specified?
Level Inheritance

• The *inherited level* for a given logger $C$, is equal to the first non-null level in the logger hierarchy, starting at $C$ and proceeding upwards in the hierarchy towards the root logger.

• Draw a picture and explain
Conditions of Successful Logging

- Using the logging methods does not guarantee publishing the info
- **Logging messages with level less than logger level will be ignored**: Any logging information will be approved by a logger if and only if the level $p$ associated with the logging message is greater than or equal to the level $q$ assigned to the logger
- The Appender objects can use Filter and further reject—-but this does not apply to us
Boilerplate code for writing to file

- `log4j.appender.file`

  org.apache.log4j.FileAppender
- `log4j.appender.stdout.Target=hibernate.log`
- `log4j.appender.stdout.layout=org.apache.log4j.PatternLayout`
- `log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n`

Then we can write:
- `log4j.rootLogger=info,file`
The