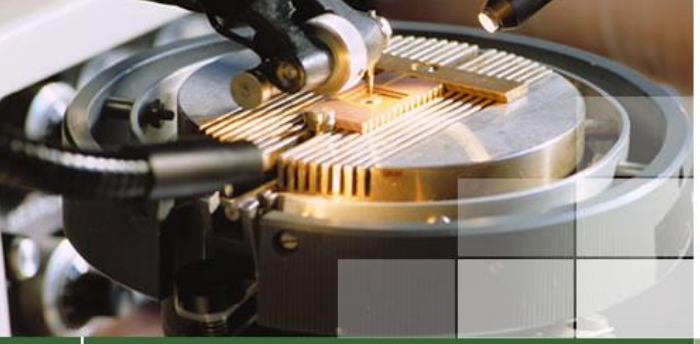


CEFRIEL
FORGING INNOVATION POLITECNICO DI MILANO

POLITECNICO DI MILANO

Dipartimento di
Elettronica e Informazione

Planning and Managing Software Projects 2011-12

Logging frameworks

Emanuele Della Valle, Lecturer: Daniele Dell'Aglio
http://emanueledellavalle.org

Outline

- Logging framework
- Key concepts
  - Declaration and Naming
  - Levels
  - Appenders
  - Layouts
- Java logging frameworks
  - Logging systems: **Log4J**, JUL, LogBack
  - Logging systems façades: ACL, **SLF4J**

Why a logging framework?

- System.out/err are not enough!
- Logging frameworks overcome the System.out/err limits
  - Several importance levels for messages
  - Use of different output systems (console, file, mail...)
  - ...
- We will consider Log4J as logging framework (but the concepts are similar in other systems)

Example (example1 package)

```
...
public int sum(int a, int b){
      System.out.println(
            "The input values are "+a+" and "+b);
      int ret = a+b;
      System.out.println("The sum is "+ret);
      return ret;
}
...
```

Example

```
...
Logger logger = Logger.getLogger(Example.class);
public int sum(int a, int b){
      logger.info(
            "The input values are "+a+" and "+b);
      //System.out.println(
            "The input values are "+a+" and "+b);
      int ret = a+b;
      logger.info("The sum is "+ret);
      //System.out.println("The sum is "+ret);
      return ret;
}
...
```

Declaration and Naming

- Loggers are initializated through a factory method

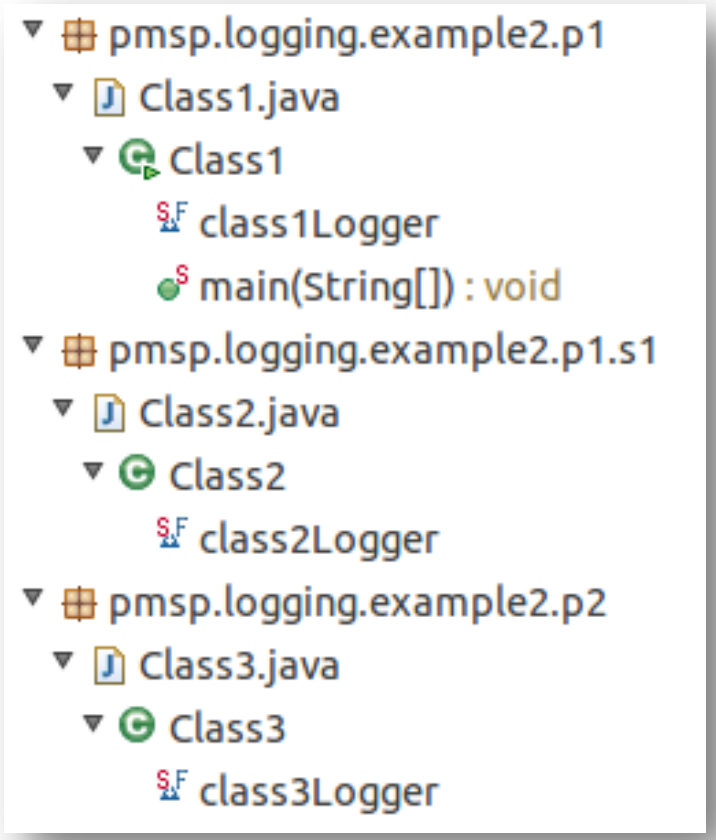
```
Logger x = Logger.getLogger("Logger1");
```

- The logger name is unique
  - Two invocations of “Logger1” return the same logger object
- Usually logger are named with the name of the class

```
Logger x = Logger.getLogger(ClassX.class);
```

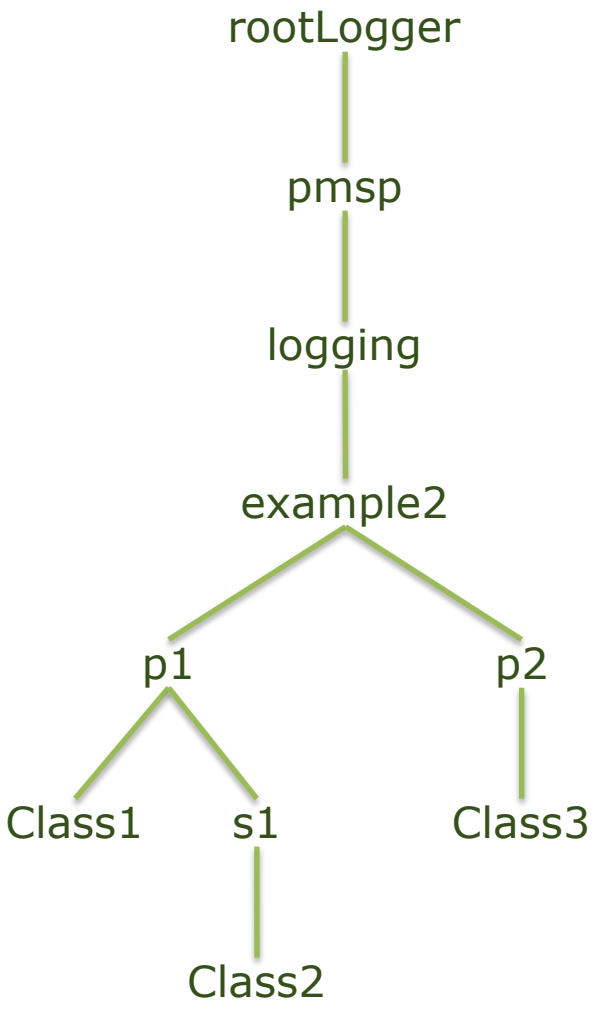
- Log4J builds and manages a tree of loggers
  - Nodes and leaves are determined through logger names
  - The root of the tree is called rootLogger
  - The tree is a key concept for the management of the logger system (as we will see in the next slides)

Example (example2 package)



Where each logger is declared as:

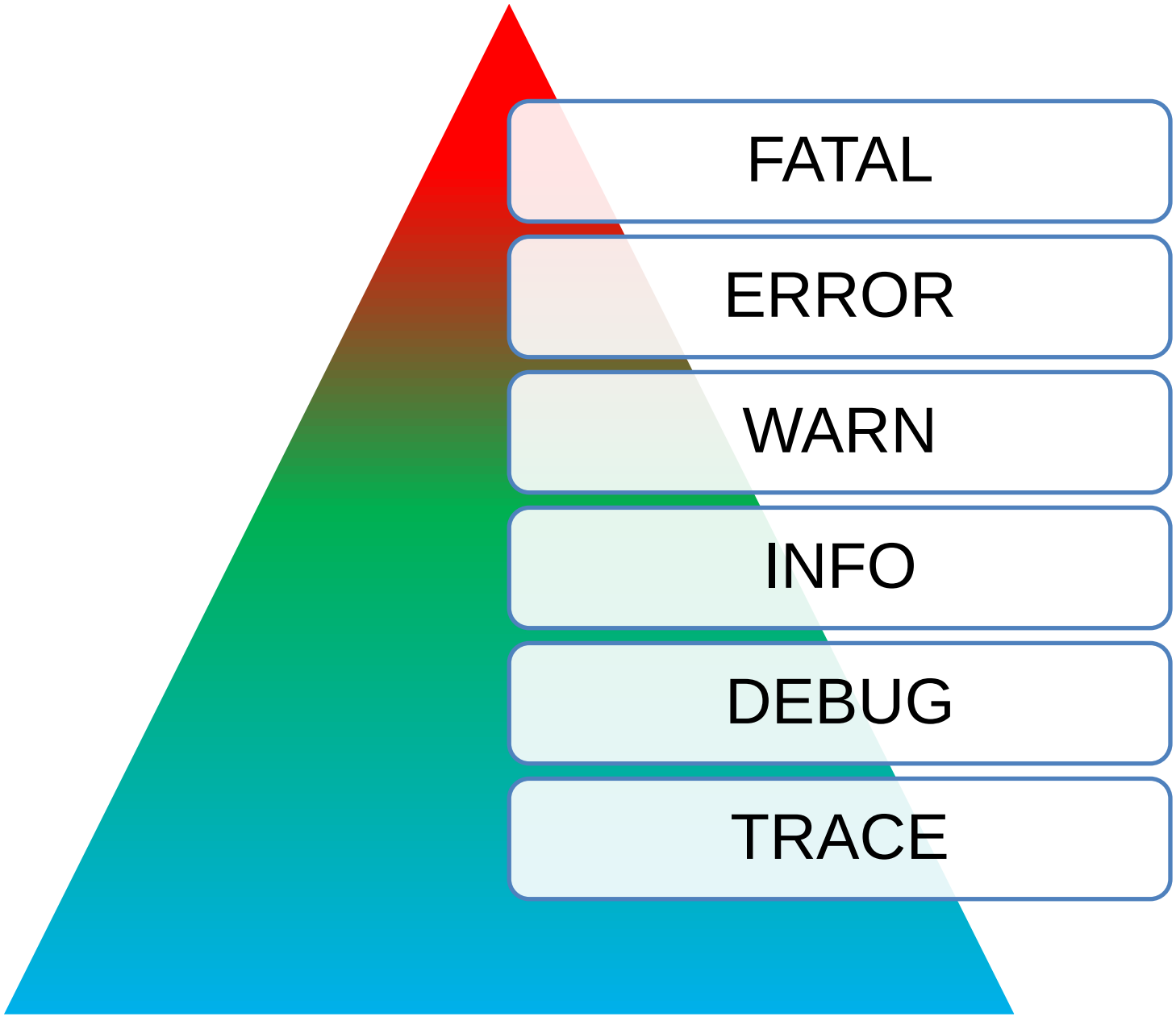
```
Logger classXLogger=
  Logger.getLogger(ClassX.class)
```



Levels

- While developing a system is useful to print out messages to check the correct behaviour of the application
- When the application is deployed most of those messages are not relevant
- Logging frameworks allow to define several levels of importance for the messages

Level hierarchy in Log4J



Meaning of the levels

- There are not strictly rules to verify if levels are used correctly
- Anyway the Log4j documentation indicates a general guide line:
  - FATAL: designates very **severe error** events that will presumably lead the application to **abort**.
  - ERROR: designates **error** events that might still allow the application to **continue running**.
  - WARN: The WARN level designates potentially **harmful situations**.
  - INFO: designates informational messages that highlight the **progress of the application** at coarse-grained level.
  - DEBUG: designates **fine-grained informational** events that are most useful to **debug** an application.
  - TRACE: designates **finer-grained informational** events than the DEBUG

Configuring the levels

- Each logger has an assigned level. The level is assigned in to possible ways
- Explicit declaration: the level associated to a logger is declared:
  - in the configuration file:

```
log4j.logger.<logger-name>=<level>
```

  - or in the code:

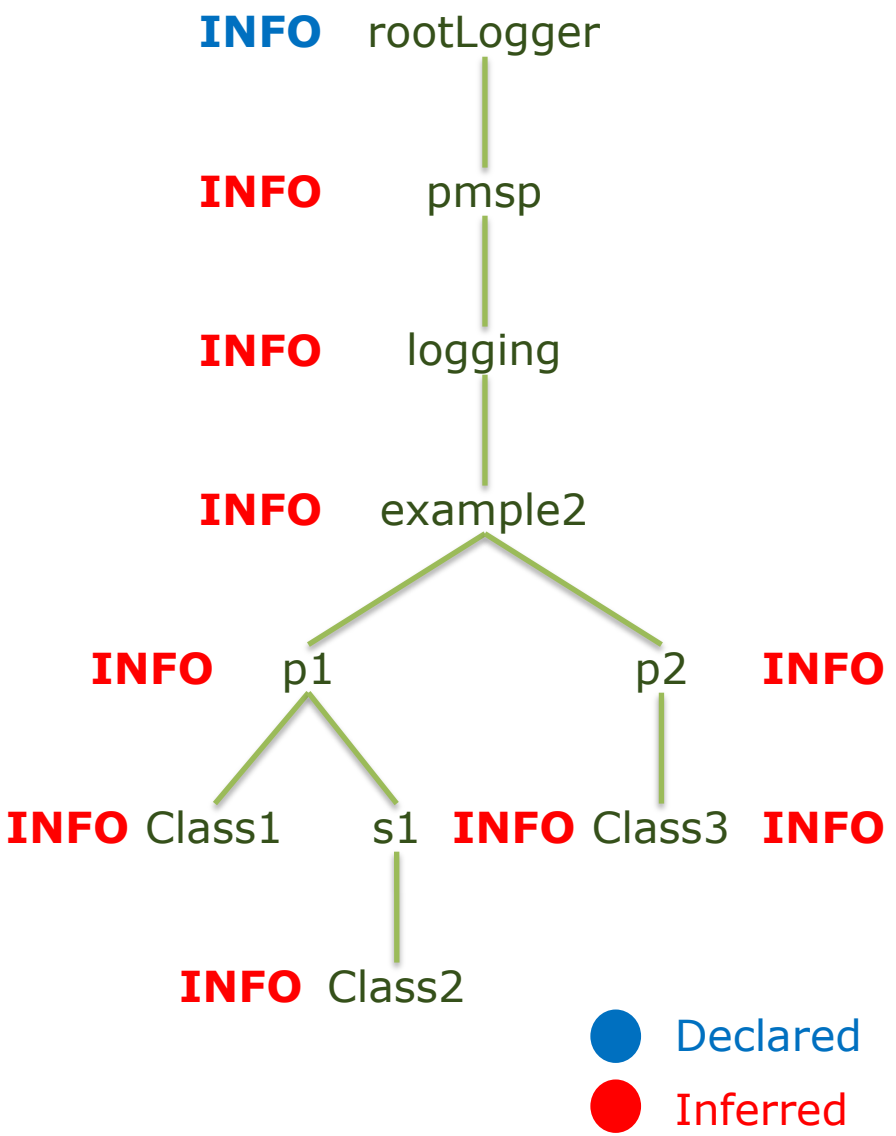
```
logger.setLevel(Level.<level>);
```

- Implicit declaration: the level is inferred through the logger tree
  - Loggers inherit the level of their parents

Example

- In the properties file:

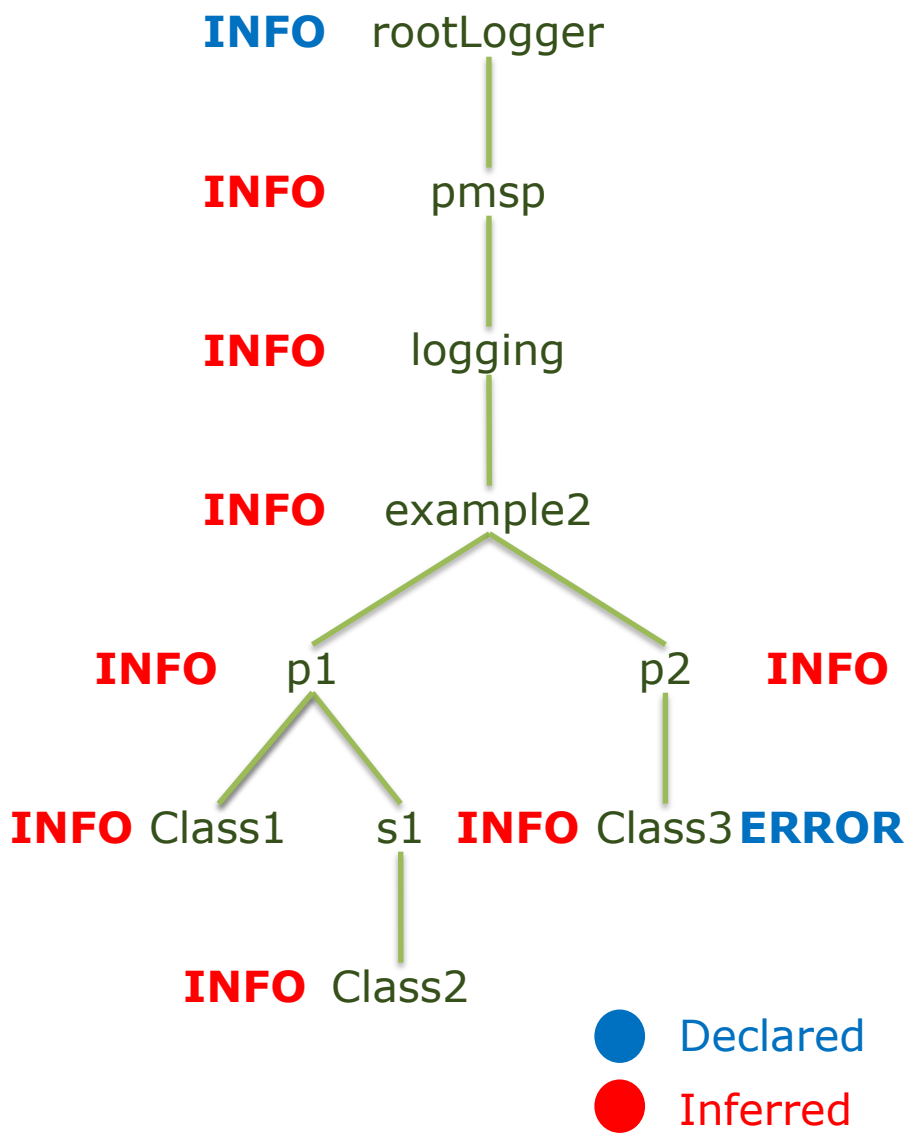
```
log4j.rootlogger=INFO
```



Example

- In the properties file:

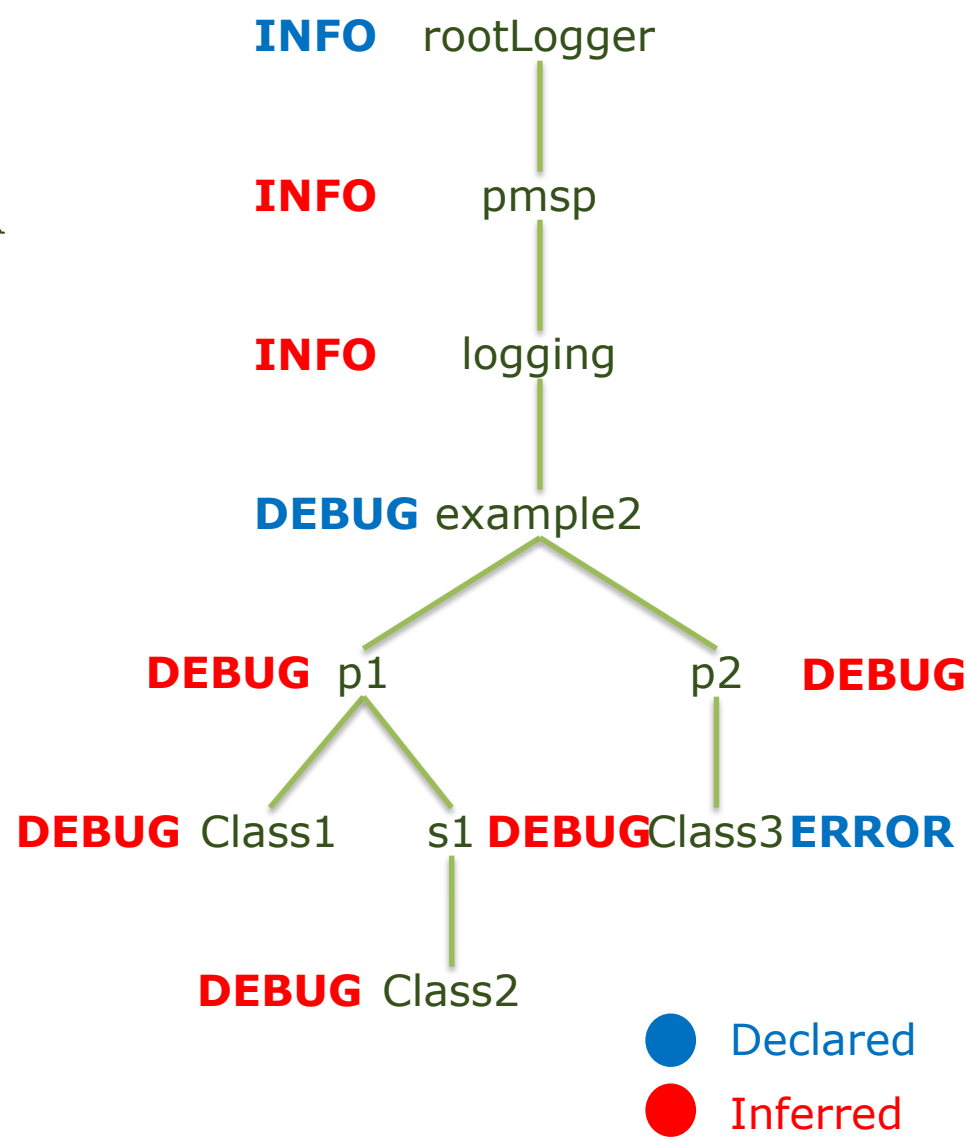
```
log4j.rootlogger=INFO
log4j.logger.pmsp.logging.
      example2.p2.Class3=ERROR
```



Example

- In the properties file:

```
log4j.rootlogger=INFO
log4j.logger.pmsp.logging.
      example2.p2.Class3=ERROR
log4j.logger.pmsp.logging.
      example2=DEBUG
```



Appenders

- Loggers can write on several output systems
- Each output system is defined by an appender
- There is a high number of available appenders in Log4J
  - Console, File, DB, network, etc.
  - Appenders are declared and configured in the properties file

```
log4j.appenders.<name>=<appender class>
log4j.appenders.<name>.<param1>=...
log4j.appenders.<name>.<param2>=...
```

- Each logger is associated to one or more appenders
  - The assignment is managed through the logger tree (like the levels)

Example

```
#Rolling file appender: fileap
log4j.appender.fileap=org.apache.log4j.RollingFileAppender
log4j.appender.fileap.File=log.xml
log4j.appender.fileap.MaxFileSize=100KB

#Stdout appender: stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.rootLogger=INFO,stdout
log4j.logger.psmp.logging.example2.p2.Class3=FATAL,fileap
```

(Class3 is associated to both stdout and fileap appenders)

More info is availabe in the Log4j documentation:
http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/AppenderSkeleton.html

Layouts

- Layouts defines the format of the log lines
- Each appender is related to one layout

```
log4j.appender.<name>.layout=<layout class>
log4j.appender.<name>.layout.<param1>=...
log4j.appender.<name>.layout.<param2>=...
```

- Some layouts built-in in Log4J are:
  - DateLayout
  - HTMLLayout
  - XMLLayout
  - PatternLayout

Example

```
#Rolling file appender
log4j.appender.fileap=org.apache.log4j.RollingFileAppender
log4j.appender.fileap.File=log.xml
log4j.appender.fileap.MaxFileSize=100KB
log4j.appender.fileap.layout=
      org.apache.log4j.xml.XMLLayout
#Stdout appender
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=
      org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=
       %d [%t] %-5p %c - %m%n
```

Logging frameworks in Java

- Log4J is not the only available logging framework for Java
- Other notable logging systems are:
  - Java Util Logging: built-in in the Java framework, it can be used without including external libraries in the project. On the other hand, it has a complex level hierarchy and less features than Log4J
  - LogBack: aims to be the succesor of Log4j. This framework is faster than Log4J and overcomes some of its limits

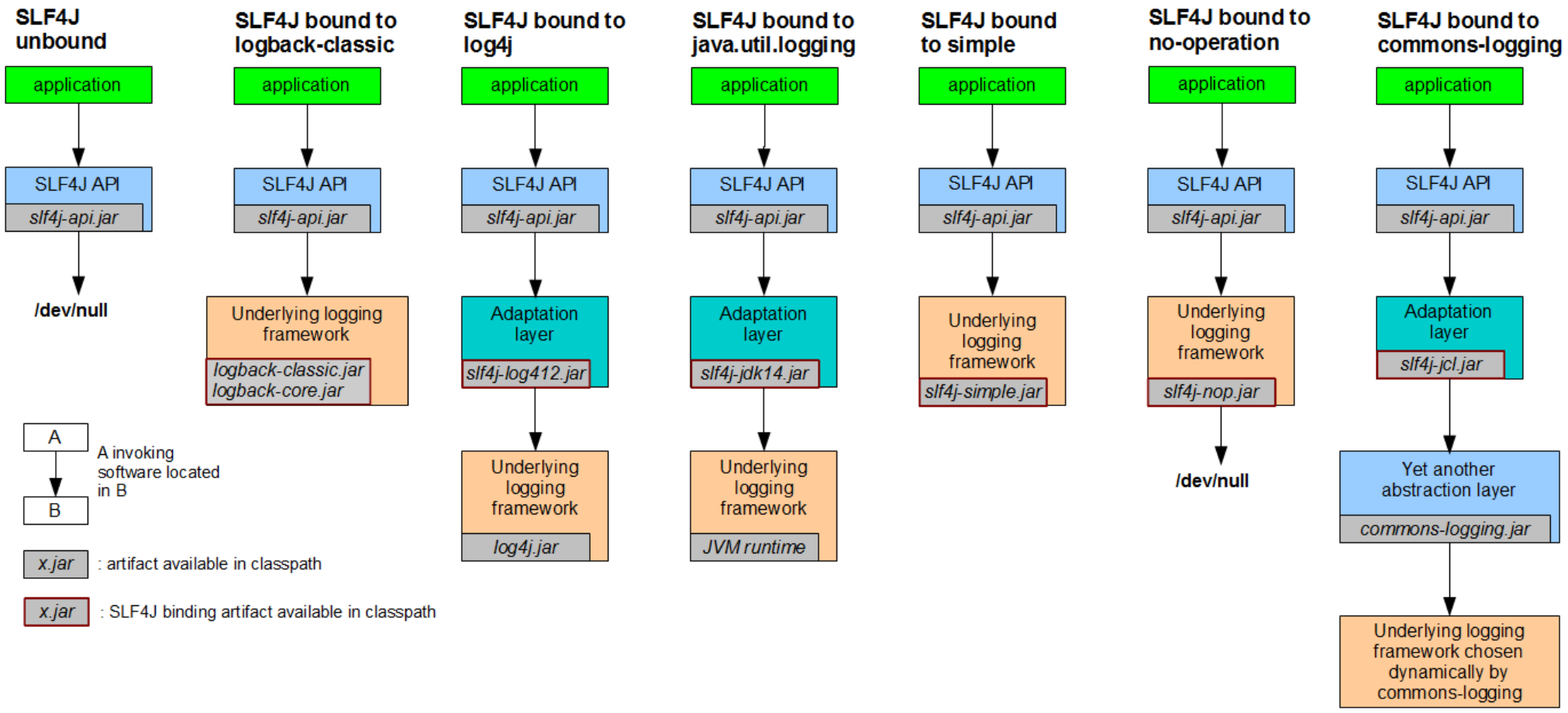
Logging façades

- It is considered a bad practice to create a strong dependance between the code and a specific logging framework
- The best practice is to use a logging façade that "wraps" the logging framework system
- The two most used systems in Java are
  - Apache Commons Logging (formerly Jakarta Commons Logging)
  - SLF4J

SLF4J

- SLF4J abstracts the logging concepts allowing users to decide what logging system use in their application
- SLF4J is “configured” at deploy time
  - Developer should import the right libraries into the project
  - Classpath must have at most one bridge library

SLF4J

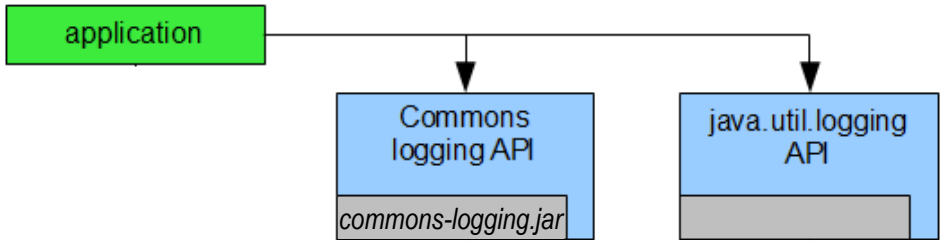


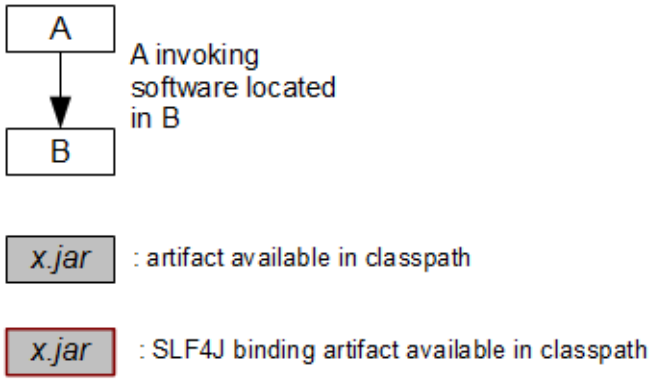
From http://www.slf4j.org/manual.html

SLF4J

- In huge projects several logger framework could be used
- In this case SLF4J allows to use migration libraries
  - They bounds an existing logging framework into SLF4j

SLF4J bound to log4j with redirection of commons-logging and jul calls to SLF4J



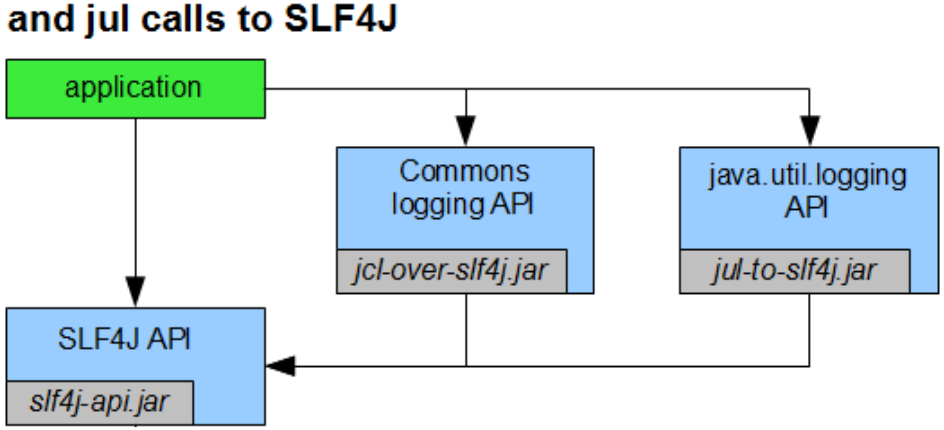


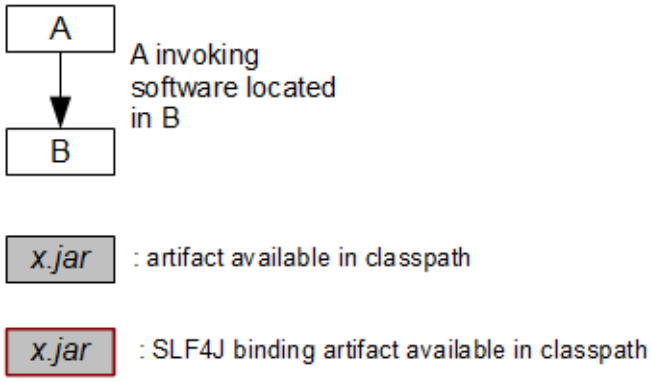
From: http://www.slf4j.org/legacy.html

SLF4J

- In huge projects several logger framework could be used
- In this case SLF4J allows to use migration libraries
  - They bounds an existing logging framework into SLF4j

SLF4J bound to log4j with redirection of commons-logging and jul calls to SLF4J



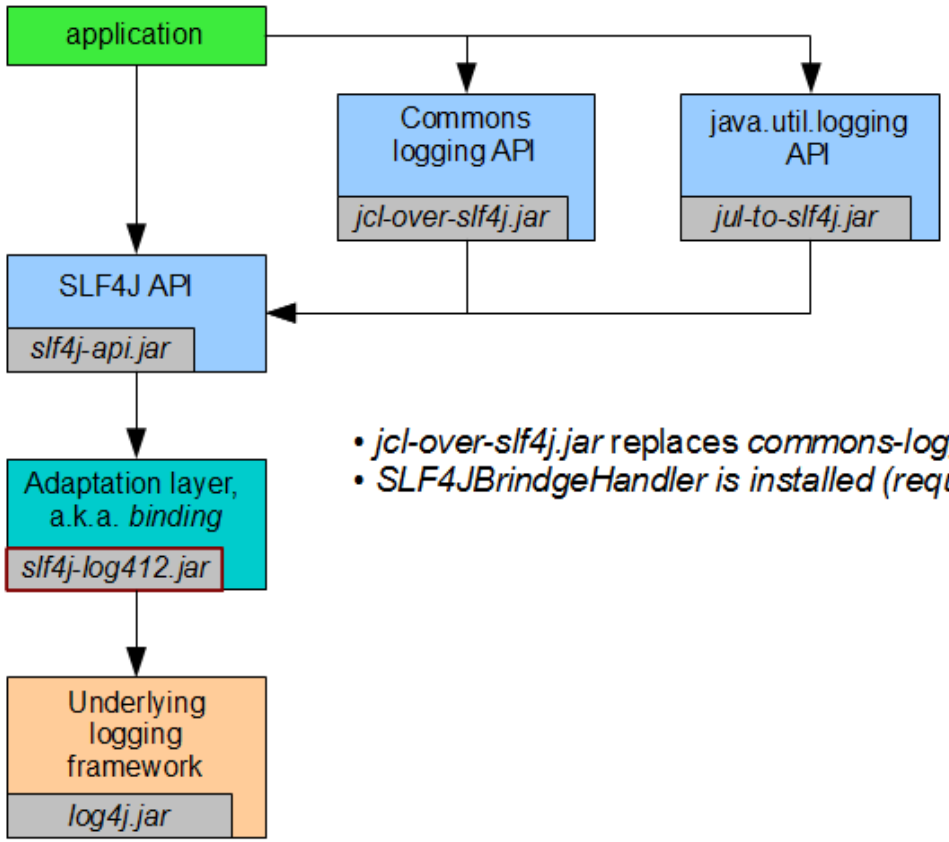


From: http://www.slf4j.org/legacy.html

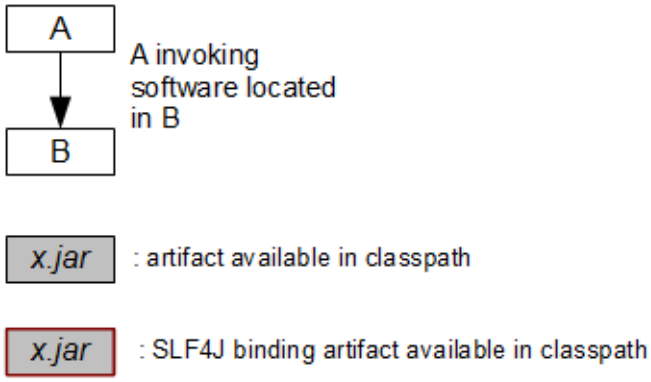
SLF4J

- In huge projects several logger framework could be used
- In this case SLF4J allows to use migration libraries
  - They bounds an existing logging framework into SLF4j

SLF4J bound to log4j with redirection of commons-logging and jul calls to SLF4J



- *jcl-over-slf4j.jar* replaces *commons-logging.jar*
- *SLF4JBrindgeHandler is installed (requires jul-to-slf4j.jar)*



From: http://www.slf4j.org/legacy.html

References

- Don't Use System.out.println! Use Log4j - http://www.vipan.com/htdocs/log4jhelp.html
- Apache Log4J manual http://logging.apache.org/log4j/1.2/manual.html
- SLF4J manual http://www.slf4j.org/manual.html
- Think again before adopting the commons-logging API http://articles.qos.ch/thinkAgain.html
- Logging, which framework to choose? Log4j, Commons Logging, LogBack, SLF4J http://mike.vanvendeloo.net/2011/05/06/logging-which-framework-to-choose-log4j-commons-logging-logback-slf4j