Logging

Using log4j, JCL and SLF4J

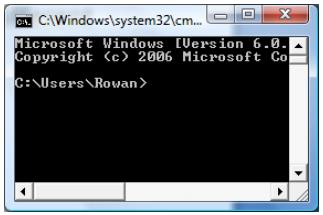
EHSDI

eBuzima

Why log?

- In any large application, logging is necessary to monitor..
  - What the application is doing
  - Why errors might be occurring
- It is not always possible or practical to debug
- When a non-programmer finds an error, they can submit information from logs to the developers

System.out.println?



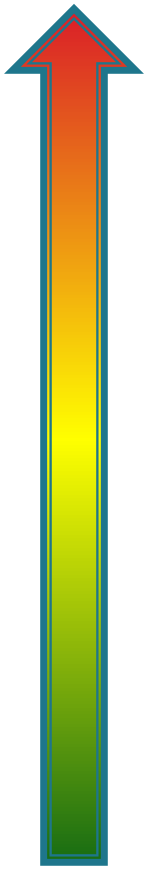
- If you have a small application, then a few `println` calls may suffice
- But for a larger application we might require:
  - Keeping a record of log messages for each day
  - A way of distinguishing between serious errors and just useful information
  - The ability to send log messages to different places, e.g. the console, a file, email

Log messages

- To be useful, log messages should contain the following information:
  - The severity level (e.g. ERROR or WARN)
  - The date and time
  - The source – usually the Java class name
  - The message – some useful information for the user or developer

Log levels

Name	Description
FATAL[1]	Severe errors that cause premature termination
ERROR	Other runtime errors or unexpected conditions
WARN	Other runtime situations that are undesirable or unexpected, but not necessarily "wrong"
INFO	Interesting runtime events (startup/shutdown)
DEBUG	Detailed information on the flow through the system
TRACE	Even more detailed information



[1] – not used by SLF4J

Java logging libraries

- `java.util.logging` (JUL)
  - Part of the JDK 1.4+
- log4j
  - Part of the Apache logging services
  - More features than JUL
  - Widely used by open source community, including OpenMRS

Logging facades

- Facades are designed to work with many different logging libraries, and provide a common interface
- This means the application's logging calls are independent of the logging library used

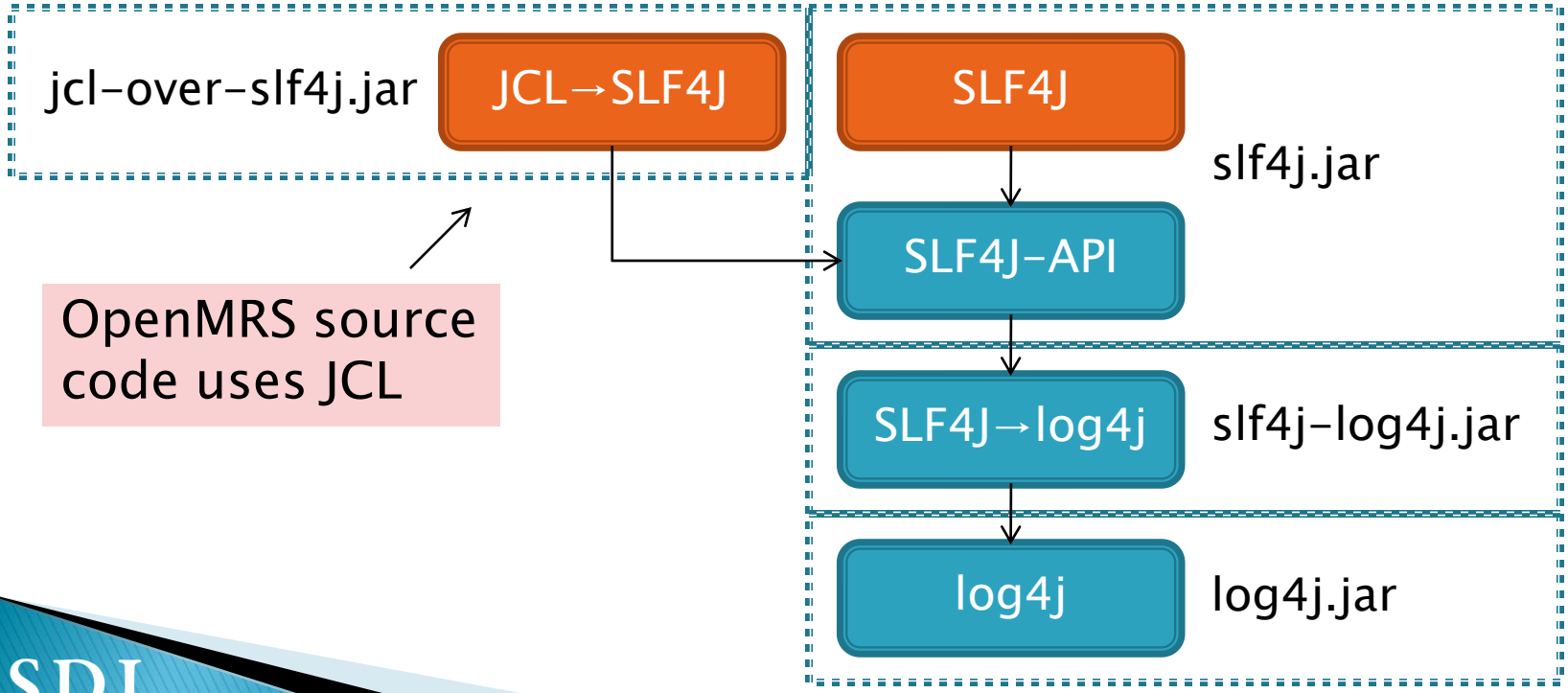
- Most commonly used ones are:
  - Jakarta Commons Logging (JCL)
  - Simple Logging Façade for Java (SLF4J)

log4j + SLF4J + JCL

- Different libraries within an application may use different logging frameworks
- To ensure that all logging messages end up in the same place, we can use bridges
- For example:
  - `jcl-over-slf4j.jar` replaces the JCL JAR file, and redirects all JCL log messages to SLF4J

log4j + SLF4J + JCL

- For an application like OpenMRS, that needs to support both JCL and SLF4J interfaces, then it can be configured as follows:



Jakarta Commons Logging (JCL)

- Used as the logging **frontend** in OpenMRS
- For each log level, there is a corresponding output method
- Loggers are created using `LogFactory`
- For example:

```
fatal(...)
error(...)
warn(...)
info(...)
debug(...)
trace(...)
```

```
public class HelloWorld {
  public static void main(String[] args) {
    Log log = LogFactory.getLog(HelloWorld.class);
    log.info("Hello World");
  }
}
```

Logging for Java (log4j)

- Used as the logging **backend** in OpenMRS
- Configurable using a properties file or XML file
- For example:

```
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
  <appender name="CONSOLE" class="org.apache.log4j.ConsoleAppender">
    <layout class="org.apache.log4j.SimpleLayout" />
  </appender>

  <logger name="org.openmrs">
    <level value="WARN"/>
  </logger>

  <root>
    <appender-ref ref="CONSOLE" />
  </root>

</log4j:configuration>
```

Equivalent to a log in JCL

All loggers inherit from root

Example: Logs and Loggers

- In the class: `org.openmrs.web.HelloWorld`

```
Log log = LogFactory.getLog(HelloWorld.class);
```

- This creates a JCL log with the name *"org.openmrs.web.HelloWorld"*
- This log will inherit properties from the following log4j loggers (if they are defined in log4j.xml)
  - root
  - org
  - org.openmrs
  - org.openmrs.web

More specific loggers override less specific

Logger levels

- A logger is configured with a log level
- All log messages with that level or above will be recorded
- All log messages below that level will be ignored
- For example:

```
<logger name="org.openmrs">
  <level value="WARN"/>
</logger>
```

Will output:
FATAL
ERROR
WARN

Will ignore:
INFO
DEBUG
TRACE

Appenders

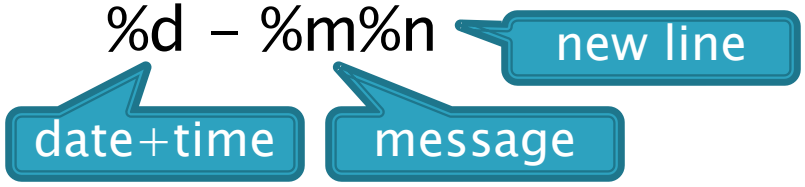
- Output of the log messages is done by *appenders*
- More than one appender can be attached to a logger, so that the log messages can be outputted in more than one way
- Many appenders exist:
  - FileAppender, ConsoleAppender, SocketAppender, SyslogAppender, NTEventLogAppender, SMTPAppender, JDBCAppender, and more

Layouts

- These are responsible for formatting the log messages
- There are several to choose from
  - **SimpleLayout**
    - `DEBUG - This is a log message`
  - **HTMLLayout**
    - `<tr><td>DEBUG</td> .... </tr>`
  - **XMLLayout**
    - `<log4j:eventSet>...</log4j:eventSet>`

Pattern Layout

- This is the most flexible layout
- Slightly slower which may be important in applications where performance is important
- Format is completely customizable using a conversion pattern, e.g.
  - %d - %m%n



See http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html

Configuration

- log4j can be configured using one of
  - log4j.properties (old format)
  - log4j.xml (new XML format)
- log4j searches for a file with one of these names on the classpath
- Thus for a web app, the file can be placed in `/WEB-INF/classes`

Example

```
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

  <appender name="CONSOLE" class="org.apache.log4j.ConsoleAppender">
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern"
             value="%p - %C{1}.%M(%L) |%d{ISO8601}| %m%n" />
    </layout>
  </appender>


  <logger name="org.openmrs">
    <level value="INFO" />
  </logger>


  <root>
    <level value="WARN" />
    <appender-ref ref="CONSOLE" />
  </root>

</ log4j:configuration>
```

Order of elements is important:

1. Appenders
2. Loggers
3. Root

References

- Websites
  - http://logging.apache.org/log4j
  - http://www.slf4j.org/