Logging needs

- Use logging as part of our debugging process
- Use logging to monitor the status of a live application
- Any web application with more than a few screens will have some kind of process or data that needs to be monitored.
- A good logging technology should take care of both of these aspects in one simple, lightweight solution
- What makes a good logging technology???

What makes a good logging technology???

- **Flexibility of output**
  - A good logging technology allows for logging messages to be output to a variety of destinations and to more than one destination at a time
- **Different levels of logging**
  - A good logging solution can differentiate between message levels and allow for log messages for each level to be selectively turned on or off without touching the source code of the application

What makes a good logging technology???

- **Performance**
  Any logging solution we choose to employ should not affect the performance of the application

- **loosely coupled**
  it's ability to stay loosely coupled to the application itself

Apache log4j is a logging framework which satisfies above points related to a good logging technology

Three main parts of log4j:

- public class Logger

  Logger is responsible for handling the majority of log operations.

- public interface Appender

  Appender is responsible for controlling the output of log operations.

- public abstract class Layout

  Layout is responsible for formatting the output for Appender.

Log4J Levels:

In log4j, there are 5 normal levels :

- ***Level DEBUG***
  - Designates fine-grained informational events that are most useful to debug an application.
- ***Level INFO***
  - Gives general information about the application such as startup and shutdown messages

Log4J Levels:

- ***Level WARN***
  - Designates potentially harmful situations, that are undesirable but not necessarily an error. This can be useful for flagging issues related to performance and security
- ***Level ERROR***
  - Designates error events that might still allow the application to continue running.
- ***Level FATAL***
  - The FATAL level designates very severe error events that will presumably lead the application to abort.

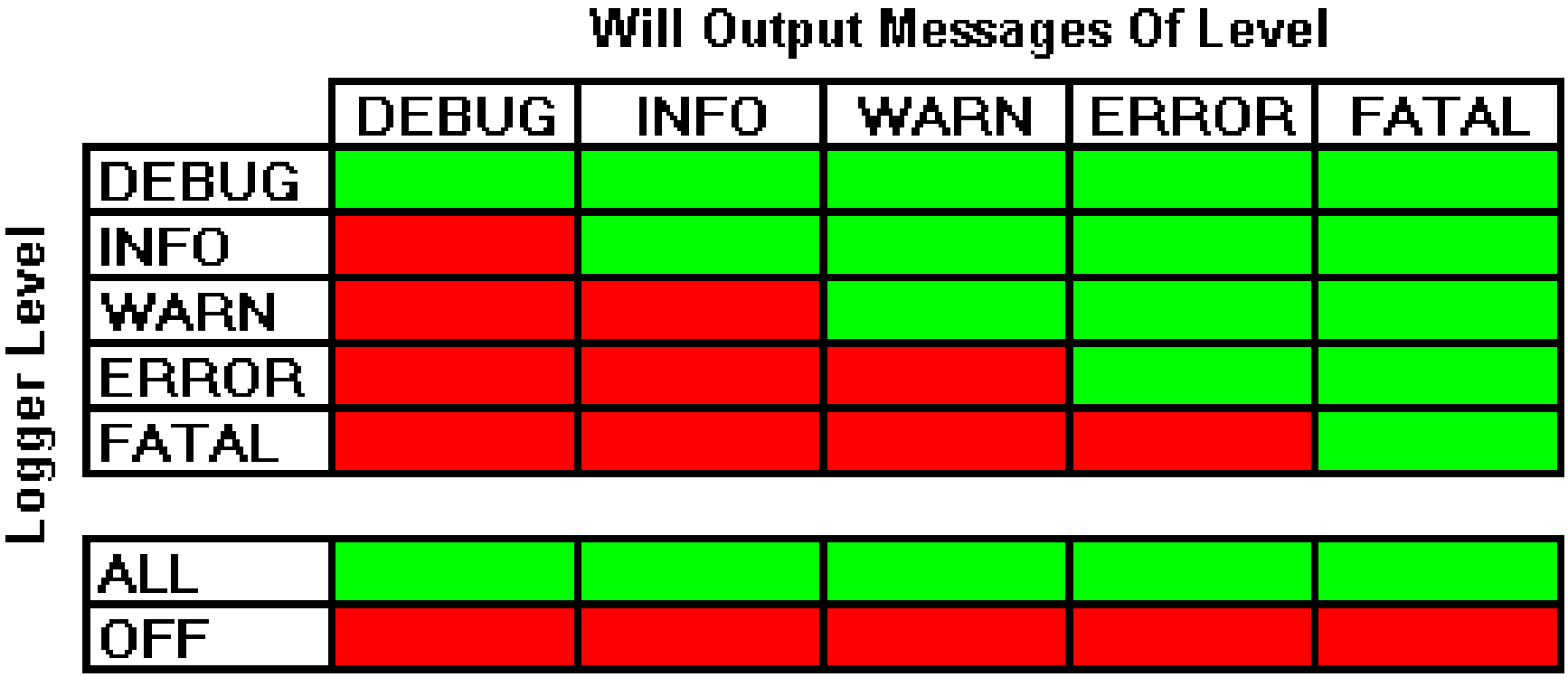
Log4J Levels:

In addition, there are two special levels of logging available:

- ***Level ALL***
  - is intended to turn on all logging.

- ***Level OFF***
  - is intended to turn off logging.

Effective use of these message levels will allow us to filter out different messages into different log destinations and selectively activate and deactivate certain log output


Will Output Messages Of Level
DEBUG
INFO
WARN
ERROR
FATAL
Logger Level
DEBUG
INFO
WARN
ERROR
FATAL
ALL
OFF

- A logger will only output messages that are of a level greater than or equal to it.
- If the level of a logger is not set it will inherit the level of the closest ancestor.
  - If a logger is created in the package com.foo.bar and no level is set for it, it will inherit the level of the logger created in com.foo.

- If no logger was created in com.foo, the logger created in com.foo.bar will inherit the level of the root logger.
- The root logger is always instantiated and available, the root logger is assigned the level DEBUG.

Different ways to create a logger

- Retrieve the root logger:

  Logger logger = Logger.getRootLogger();
- Create a new logger:

  Logger logger = Logger.getLogger("MyLogger");
- Instantiate a static logger globally, based on the name of the class:

  static Logger logger = Logger.getLogger(test.class);
- Set the level with:

  logger.setLevel((Level)Level.WARN);

Appender

The Appender controls the output of logging.

The Appenders available are

- ConsoleAppender

  appends log events to System.out or System.err using a layout specified by the user. The default target is System.out.

- DailyRollingFileAppender

  extends FileAppender so that the underlying file is rolled over at a user chosen frequency.

- FileAppender

  appends log events to a file.

- RollingFileAppender

  extends FileAppender to backup the log files when they reach a certain size.

Appender

- WriterAppender

  appends log events to a Writer or an OutputStream depending on the user's choice.

- SMTPAppender

  sends an e-mail when a specific logging event occurs, typically on errors or fatal errors.

- SocketAppender

  sends LoggingEvent objects to a remote log server,

Layouts

- The Appender must have an associated Layout so it knows how to format the output.
- There are three types of Layout available:
  - HTMLLayout
    - formats the output as a HTML table.
  - PatternLayout
    - formats the output based on a *conversion pattern* specified, or if none is specified, the default conversion pattern.
  - SimpleLayout
    - formats the output in a very simple manner, it prints the Level, then a dash '-' and then the log message.

Configuration

- Log4j is usually used in conjunction with external configuration files so that options do not have to be hard-coded within the software.

- The advantage of using an external configuration file is that changes can be made to the options without having to recompile the software.

- A disadvantage could be, that due to the *io* instructions used, it is slightly slower.

- There are two ways in which one can specify the external configuration file
  - A properties file
  - A XML file.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

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

  <appender name="ConsoleAppender"
     class="org.apache.log4j.ConsoleAppender">
    <layout class="org.apache.log4j.SimpleLayout"/>
  </appender>

  <root>
    <priority value ="debug" />
    <appender-ref ref="ConsoleAppender"/>
  </root>

</log4j:configuration>
```