Wer spricht denn da?

Überblick über das Logging in Java

Sebastian Hempel · 21.09.2022

Sebastian Hempel

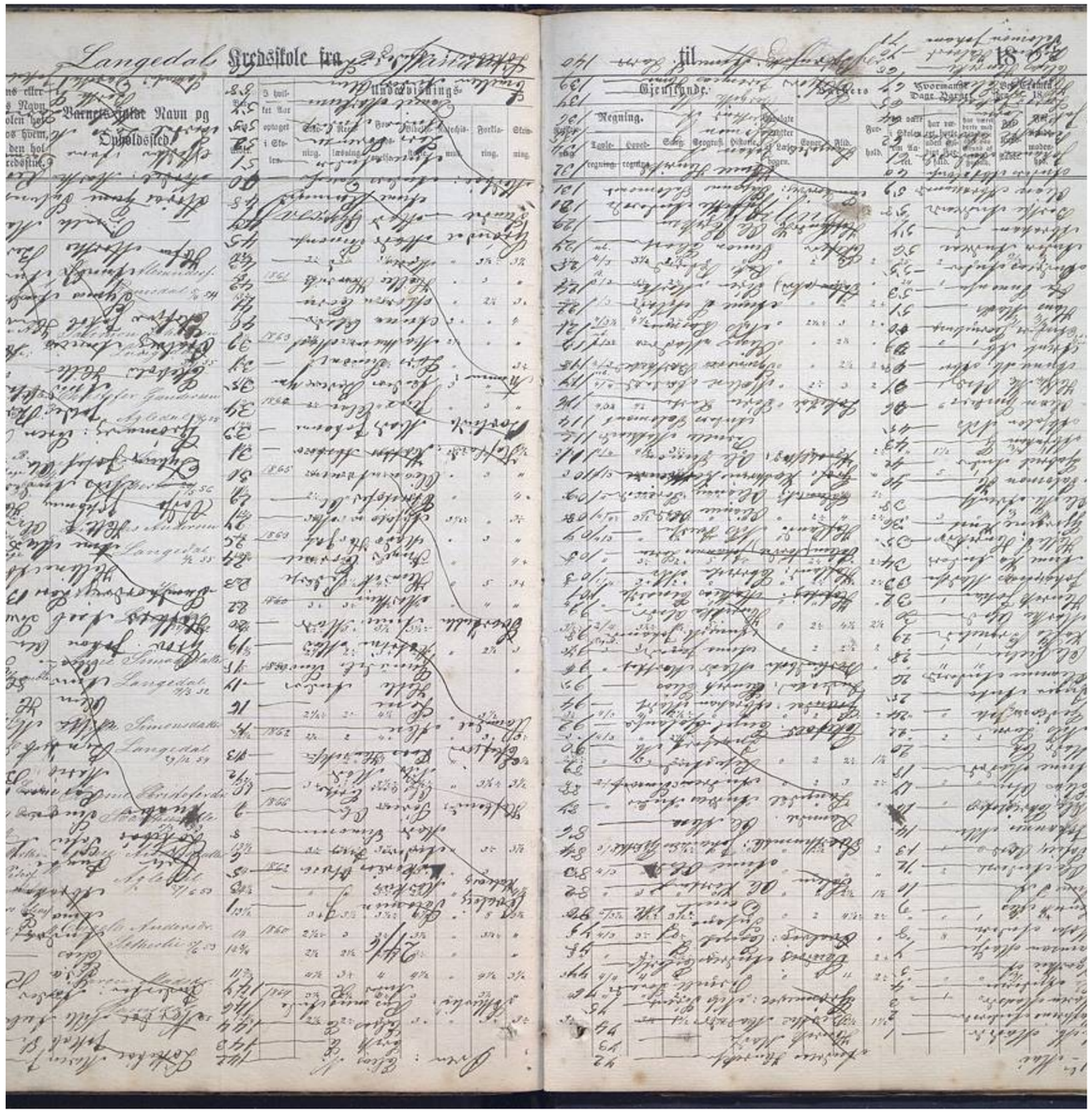
IT-Consulting Hempel

- selbständiger Software-Entwickler seit 2003
- Java, Puppet
- Linux und OpenSource
- https://it-hempel.de

Warum Logging?

Warum Logging?

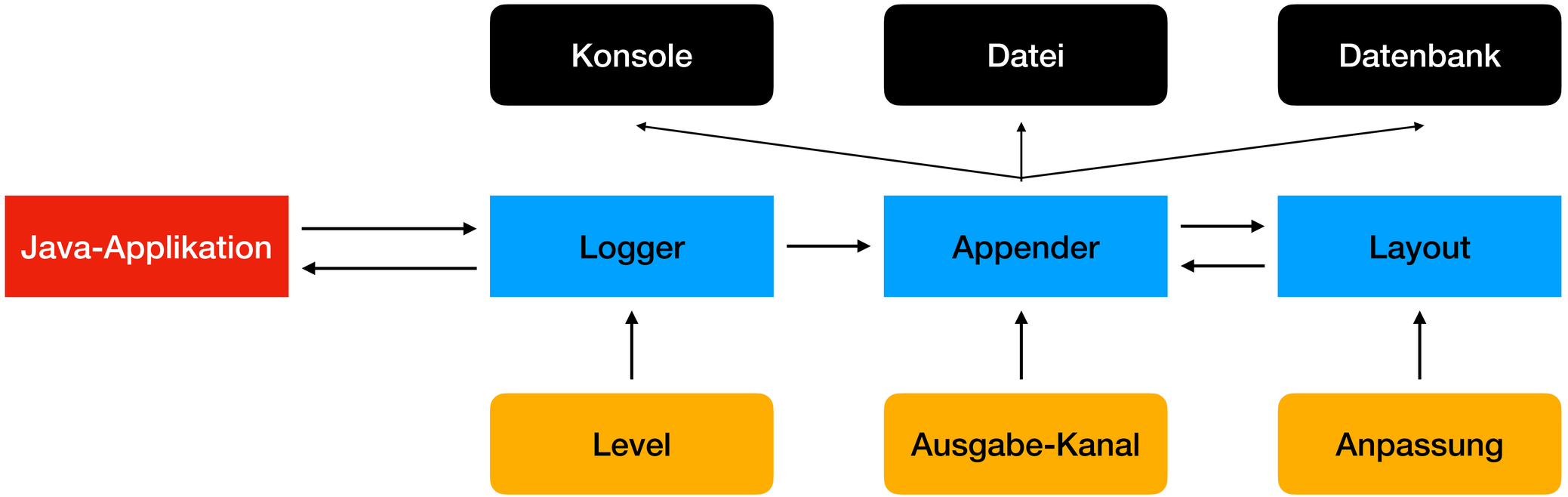
- Wie geht es meiner Applikation?
- Warum funktioniert etwas nicht?



Protokoll frå Langedal skulekrins i Flora (Skulen min) CC BY-SA

Wie funktioniert Logging?

Wie funktioniert Logging?



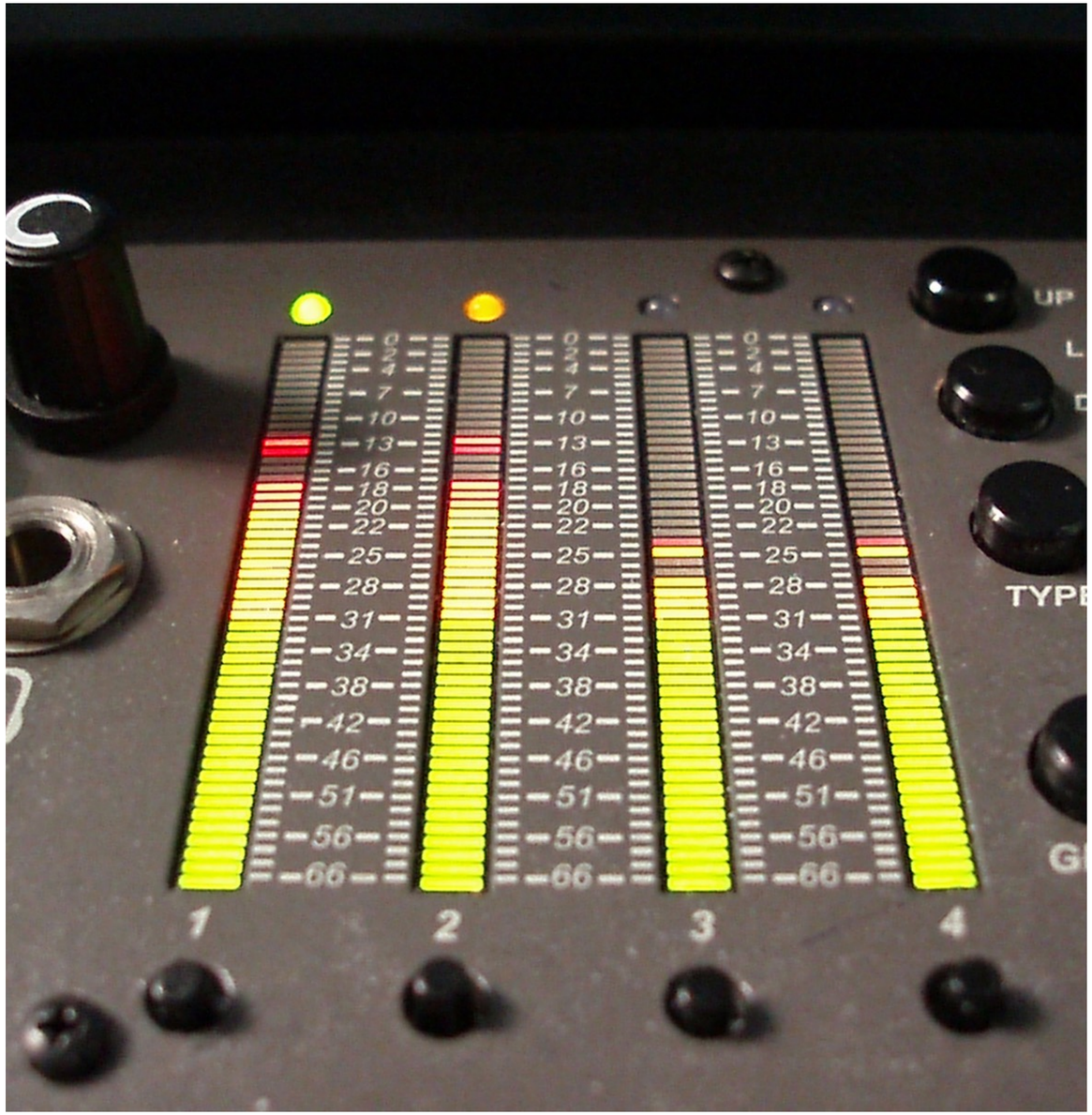
Wie funktioniert Logging?

- Logger erstellt einen LogRecord
- Appender formatiert LogRecord über Layout
- Appender gibt formatierte Meldung aus

Was sind Loglevel?

Was sind Loglevel?

- FATAL
- ERROR
- WARN
- INFO
- DEBUG
- TRACE

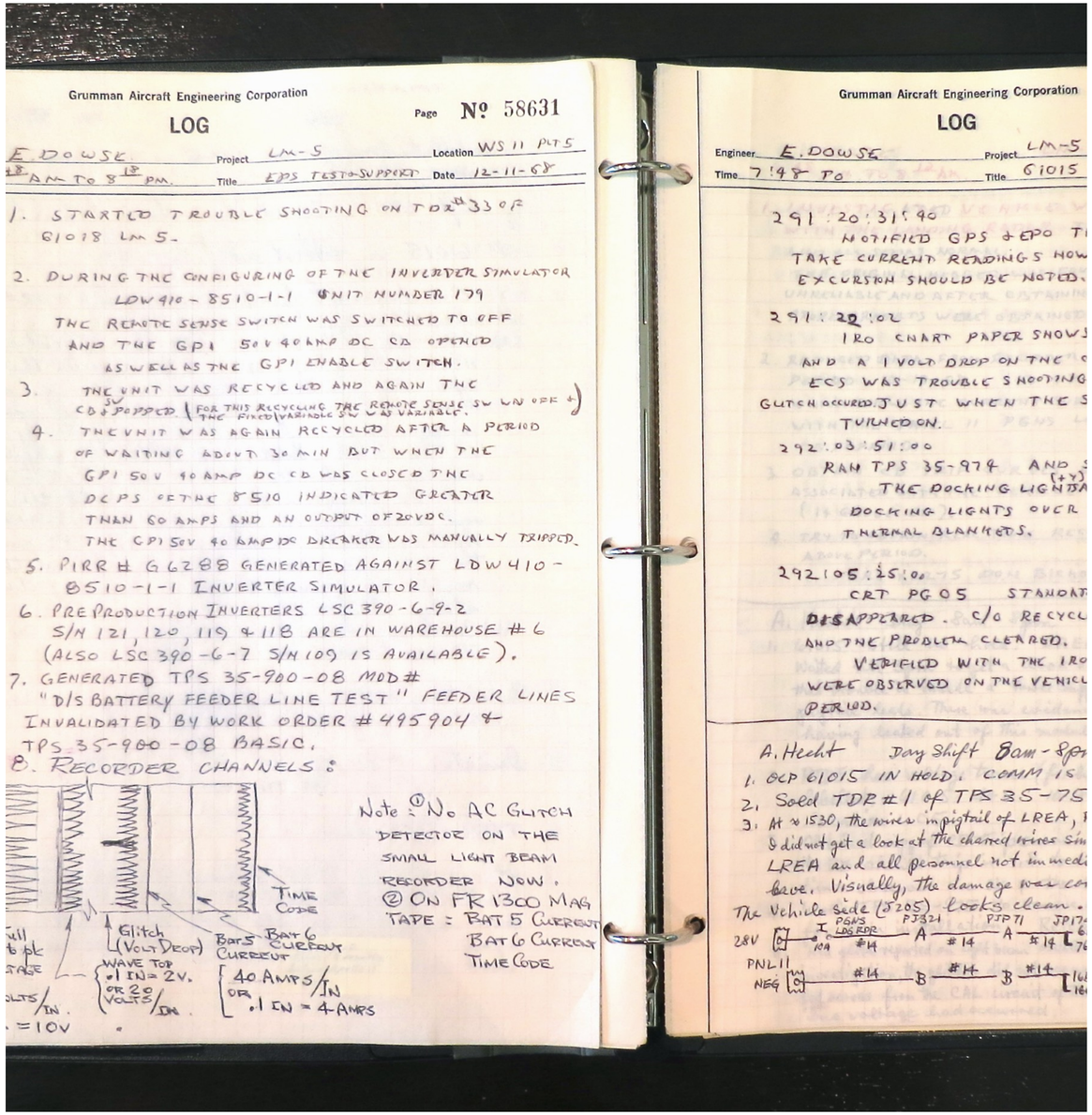


levels (Rory McSwiggan) CC BY-NC

Was beinhaltet der LogRecord noch?

Inhalt des LogRecord

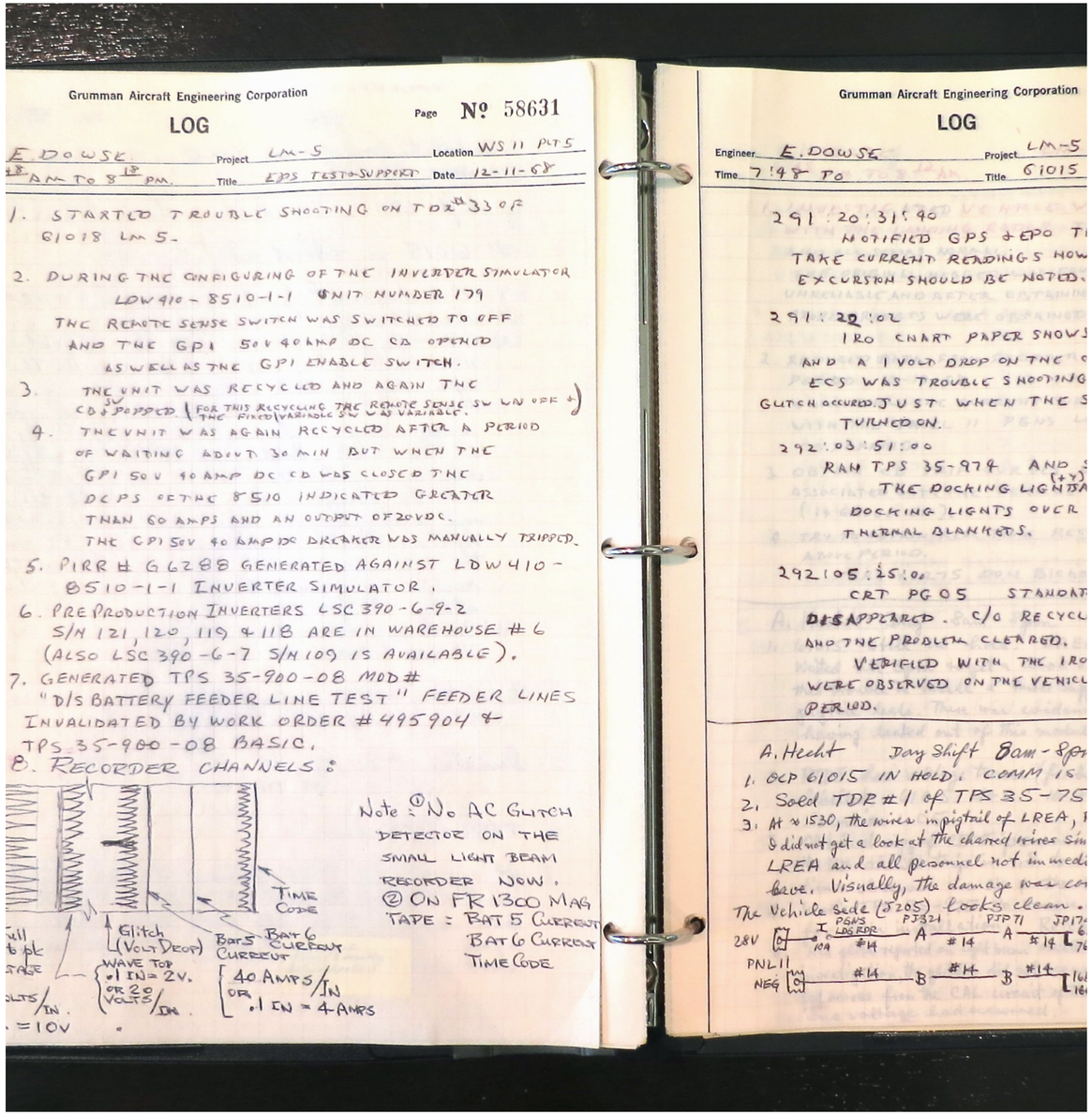
- Zeitstempel
- Logger-Name (Package und Class)
- Message
- Loglevel
- Class und Method des Aufrufes
- Source



How the Eagle Landed — the Grumman Construction Log (Steve Jurvetson) CC BY

Inhalt des LogRecord

- Thread
- NDC (nested diagnostic context)
- MDC (mapped diagnostic context)
- ThreadContext



How the Eagle Landed — the Grumman Construction Log (Steve Jurvetson) CC BY

Logger Name

- kann frei bestimmt werden
- Enthält das Package- und den Class-Name
- hierarchisch - wie Packages

Thread

- Bezeichnung des Thread / Thread-Pools
- vom Application-Server / Framework festgelegt

Context

- NDC bietet einen Stack
- MDC nutzt eine Map (key-value)
- ThreadContext nutzt eine Map
- zusätzliche, von der Anwendung gesetzte Informationen
  - UserId / UserName
  - Request
  - Tracing ID

Was sollte ich loggen?

Was wird ausgegeben?

fachlich

- (fachliche) Ereignisse der Applikation
- (unerwarteter) Zustand der Applikation
- Ergebnisse von Berechnungen
- Änderung an Daten

Was wird ausgegeben?

technisch

- Requests
- Datenbank-Zugriffe
- Start von Threads / Aktionen

Was wird ausgegeben?

from a higher level

- authentication, authorization and access
- changes
- availability (startup and shutdown)
- resources
- threats

https://coralogix.com/blog/important-things-log-application-software/

Was ist zu beachten?

Was ist zu beachten?

- Ausgabe von Personen bezogenen Daten (DSGVO)
- für die Sicherheit relevante Informationen (Benutzernamen und Kennwörter)
- Informationen zur Infrastruktur (IP-Adressen)

Security (Patio Tours) CC BY

java.util.logging (JUL)

Gibt’s da auch etwas im JDK?

Java PNG Clipart CC BY-NC

java.util.logging (JUL)

- Verfügbar seit JDK 1.4 (13.02.2002)
- keine externen Abhängigkeiten notwendig

Konfiguration

- logging.properties in $JAVA_HOME/jre/lib bis Java 8
- logging.properties in $JAVA_HOME/conf ab Java 9
- System-Property java.util.logging.config.file für Konfigurationsdatei

Nutzung


```java
package de.ithempel.javalogging;

import java.util.logging.Level;
import java.util.logging.Logger;

public class HelloWorld {

    private static Logger logger = Logger.getLogger(HelloWorld.class.getName());

    public static void main(String[] args) {
        logger.info("Hello World!");
        logger.log(Level.WARNING, "Logging at WARNING level");
    }

}
```

Nutzung


```java
package de.ithempel.javalogging;

import java.util.logging.Level;
import java.util.logging.Logger;

public class HelloWorldConfig {

    static {
        String propertiesPath = HelloWorldConfig.class.getClassLoader()
            .getResource("logging.properties").getFile();
        System.setProperty("java.util.logging.config.file", propertiesPath);
    }

    private static Logger logger = Logger.getLogger(HelloWorldConfig.class.getName());

    public static void main(String[] args) {
        logger.info("Hello World!");
        logger.log(Level.WARNING, "Logging at WARNING level");
    }

}
```

andere Libraries

- JUL kennt keine anderen Libraries
- andere Libraries aber kennen JUL

log4j / log4j2

Der Urahn aller Logger

log4j

- erste Veröffentlichung vom 08.01.2001
- Entwickelt von Ceki Gülcü bei IBM (1997)
- Übergabe an die Apache Foundation im Jahr 2000
- Apache License 2.0

Ceki Gülcü PD

log4j

- Nach „feature complete“ keine weitere aktive Entwicklung
- Keine Trennung zwischen API und Implementierung
- Fork Reload4j für Security Fixes

log4j2

- Antwort auf die Entwicklung von Slf4j / Logback durch Gülcü
- komplette Neuimplementierung durch Ralph Goers (2009)
- erste Veröffentlichung im Juli 2014
- Apache License 2.0

log4j2

- Trennt API und Implementierung
- nur Dependency auf API Artefakt
- Formatierung mittels {}
- Unterstützung von Lambdas

log4j

- asynchrone Logger
- Thread Context
- verliert keine Nachrichten beim Konfigurationswechsel wie Logback

Einbindung


```xml
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-api</artifactId>
  <version>2.18.0</version>
</dependency>
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
  <version>2.18.0</version>
</dependency>
```

Nutzung


```java
package de.ithempel.log4j;

import org.apache.logging.log4j.Level;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class HelloWorld {

    private static Logger logger = LogManager.getLogger(HelloWorld.class.getName());

    public static void main(String[] args) {
        logger.info("Hello World!");
        logger.log(Level.WARN, "Logging at WARN level");

        logger.info("Logging with logger {}", logger.getName());
    }

}
```

Nutzung


```java
package de.ithempel.log4j;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class HelloWorldLambda {

    private static Logger logger = LogManager.getLogger();

    public static void main(String[] args) {
        logger.debug("Logging with logger {}", () -> logger.getName());
    }

}
```

Konfiguration

- Dateien auf dem classpath
- Unterschiedliche Dateien für Test und Production
- Formate: Properties, yaml, json und XML
- System-Property log4j2.configurationFile zur Angabe des Pfads zur Konfigurationsdatei

Konfiguration


```xml
<?xml version="1.0" encoding="UTF-8"?>

<Configuration status="WARN">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
        </Console>
    </Appenders>

    <Loggers>
        <Logger name="de.ithempel.log4j" level="debug" additivity="false">
            <AppenderRef ref="Console"/>
        </Logger>
        <Root level="error">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```

andere Libraries

logback als Implementierung

```xml
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-to-slf4j</artifactId>
  <version>2.18.0</version>
</dependency>
```

andere Libraries

log4j2 als Implementierung von Slf4j

```
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-slf4j-impl</artifactId>
  <version>2.18.0</version>
</dependency>
```

andere Libraries

log4j2 als Implementierung von JUL

```
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-jul</artifactId>
  <version>2.18.0</version>
</dependency>
```

andere Libraries

log4j2 als Backend von log4j

```xml
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-1.2-api</artifactId>
  <version>2.18.0</version>
</dependency>
```

Slf4j / Logback

A program has to be written at least twice.

SLF4J Logo MIT License

Slf4j

- Neuimplementierung von log4j von Ceki Gülcü
- Fassade vor den eigentlichen Loggern (log4j, logback)
- Erste Veröffentlichung 08.05.2006
- MIT License

Ceki Gülcü PD

Slf4j

- Nutzung des Service Loader Mechanismus von Java
- Fluent API seit Version 2.0
- Formatierung mittels {}
- Unterstützung von Lambdas
- Mapped Diagnostic Context

Logback

- Logback als native Implementierung von Slf4j
- erste Veröffentlichung 09.02.2006
- LGPL

Einbindung


```
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.3.0-alpha13</version>
</dependency>
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-core</artifactId>
  <version>1.3.0-alpha13</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>2.0.1</version>
</dependency>
```

Nutzung


```java
package de.ithempel.slf4j;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class HelloWorld {

    private static Logger logger = LoggerFactory.getLogger(HelloWorld.class);

    public static void main(String[] args) {
        logger.info("Hello World!");
        logger.info("Logging with logger {}", logger.getName());
    }

}
```

Nutzung


```java
package de.ithempel.slf4j;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class HelloWorldFluent {

    private static Logger logger = LoggerFactory.getLogger(HelloWorldFluent.class);

    public static void main(String[] args) {
        logger.atInfo()
            .setMessage("Logging with logger {}").addArgument(() -> logger.getName())
            .log();
    }

}
```

Logback Konfiguration

- XML-Konfiguration im classpath (Logback)
- Unterschiedliche Dateien für Test und Production
- System-Property logback.configurationFile zur Angabe des Pfads zur Konfigurationsdatei
- Wird nach Änderungen automatisch neu geladen

Konfiguration


```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration>

<configuration scan="true">
  <import class="ch.qos.logback.classic.encoder.PatternLayoutEncoder"/>
  <import class="ch.qos.logback.core.ConsoleAppender"/>

  <appender name="STDOUT" class="ConsoleAppender">
    <encoder class="PatternLayoutEncoder">
      <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
    </encoder>
  </appender>

  <logger name="de.ithempel.slf4j" level="DEBUG" />

  <root level="INFO">
    <appender-ref ref="STDOUT"/>
  </root>
</configuration>
```

andere Libraries

Slf4j als Implementierung von JUL

```
<dependency>
   <groupId>org.slf4j</groupId>
   <artifactId>jul-to-slf4j</artifactId>
   <version>2.0.1</version>
 </dependency>
```

andere Libraries

Slf4j als Implementierung von log4j

```
<dependency>
   <groupId>org.slf4j</groupId>
   <artifactId>log4j-over-slf4j</artifactId>
   <version>2.0.1</version>
 </dependency>
```

JBoss Logmanager / Logging

„Wir haben dann einmal selbst ein Logging-Framework implementiert.“

JBoss Logo PD

JBoss Logging

- Logging Fassade
- Unterstützt JBoss Logmanager, Log4j, Slf4j und Logback
- erste Veröffentlichung 16.02.2011
- Apache 2.0 License

Jboss Logmanager

- Genutzt im JBoss Application Server und Quarkus
- Replacement für JUL
- erste Veröffentlichung 15.10.2010
- Apache License 2.0

Jboss Logmanager

- Außerhalb der „JBoss Welt“ keine Verbreitung
- Logging Adapter um die APIs von anderen Libraries zu unterstützen
- JUL, JBoss Logging, Slf4j, Apache Commons Logging, log4j, log4j2

Einbindung


```
<dependency>
  <groupId>org.jboss.logmanager</groupId>
  <artifactId>jboss-logmanager</artifactId>
  <version>2.1.18.Final</version>
</dependency>
<dependency>
  <groupId>org.jboss.logging</groupId>
  <artifactId>jboss-logging-spi</artifactId>
  <version>2.1.2.GA</version>
</dependency>
<dependency>
  <groupId>org.jboss.logging</groupId>
  <artifactId>jboss-logging-logmanager</artifactId>
  <version>2.2.0.CR2</version>
</dependency>
```

Nutzung


```java
package de.ithempel.jbosslogging;

import org.jboss.logging.Logger;

public class HelloWorld {

    private static Logger logger = Logger.getLogger(HelloWorld.class);

    public static void main(String[] args) {
        logger.info("Hello World!");
        logger.debugf("Logging with logger %s", logger.getName());
    }

}
```

JBoss Logmanager Konfiguration

- JUL durch JBoss Logmanager ersetzen
- -Djava.util.logging.manager=org.jboss.logmanager.LogManager
- JBoss Logger wird wie JUL konfiguriert

andere Libraries

Adapter um andere Logausgaben umzuleiten

```
<dependency>
  <groupId>org.jboss.logmanager</groupId>
  <artifactId>log4j2-jboss-logmanager</artifactId>
  <version>1.1.1.Final</version>
</dependency>
<dependency>
  <groupId>org.jboss.slf4j</groupId>
  <artifactId>slf4j-jboss-logmanager</artifactId>
  <version>1.2.0.Final</version>
</dependency>
```

Apache Commons Logging

Da gibt es natürlich auch etwas von Apache

Apache Commons Logging PD

Apache Commons Logging

commons
logging

- Fassade vor unterschiedlichen Implementierungen
- erste Veröffentlichung 13.08.2002
- Apache License 2.0
- relativ inaktiv (letzte Version von 2014)

Einbindung

commons
logging

```
<dependency>
  <groupId>commons-logging</groupId>
  <artifactId>commons-logging</artifactId>
  <version>1.2</version>
</dependency>
```

Nutzung

commons logging

```java
package de.ithempel.apachecommons;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class HelloWorld {

    private static Log logger = LogFactory.getLog(HelloWorld.class);

    public static void main(String[] args) {
        logger.info("Hello World!");
    }

}
```

Konfiguration

commons
logging

- SystemProperties
- Properties Datei in classpath (commons-logging.properties)

Tinylog

New kid on the block

Tinylog

- leichtgewichtiger Logger
- entwickelt von Martin Winandy
- Version 0.1 vom 23.01.2012
- Version 1.0 vom 01.04.2015
- Apache License 2.0

Martin Winandy PD

Tinylog

- Entstand aus Problemen mit log4j
- Logger müssen nicht erst erstellt werden
- Formatierung durch {}
- Hohe Performance bei Logausgaben
- Konfiguration über Properties
- Unterstützung von Lambdas
- Liegt inzwischen in der Version 2 vor

Einbindung

```
<dependency>
  <groupId>org.tinylog</groupId>
  <artifactId>tinylog-api</artifactId>
  <version>2.5.0</version>
</dependency>
<dependency>
  <groupId>org.tinylog</groupId>
  <artifactId>tinylog-impl</artifactId>
  <version>2.5.0</version>
</dependency>
```

Nutzung

```java
package de.ithempel.tinylog;

import org.tinylog.Logger;

public class HelloWorld {

    public static void main(String[] args) {
        Logger.info("Hello World!");
        Logger.debug("Logging from class {}", HelloWorld.class.getName());
    }

}
```

Nutzung

```java
package de.ithempel.tinylog;

import org.tinylog.Logger;

public class HelloWorldLambda {

    public static void main(String[] args) {
        Logger.debug("Logging from class {}", () → HelloWorldLambda.class.getName());
    }

}
```

tinylog Konfiguration

- SystemProperties
- Properties Datei
- Separate Dateien zur Konfiguration von PROD und TEST

Konfiguration

```
writer1        = console
writer1.level  = debug

writer2        = file
writer2.file   = log.txt
writer2.level  = info
writer2.append = true
```

andere Libraries

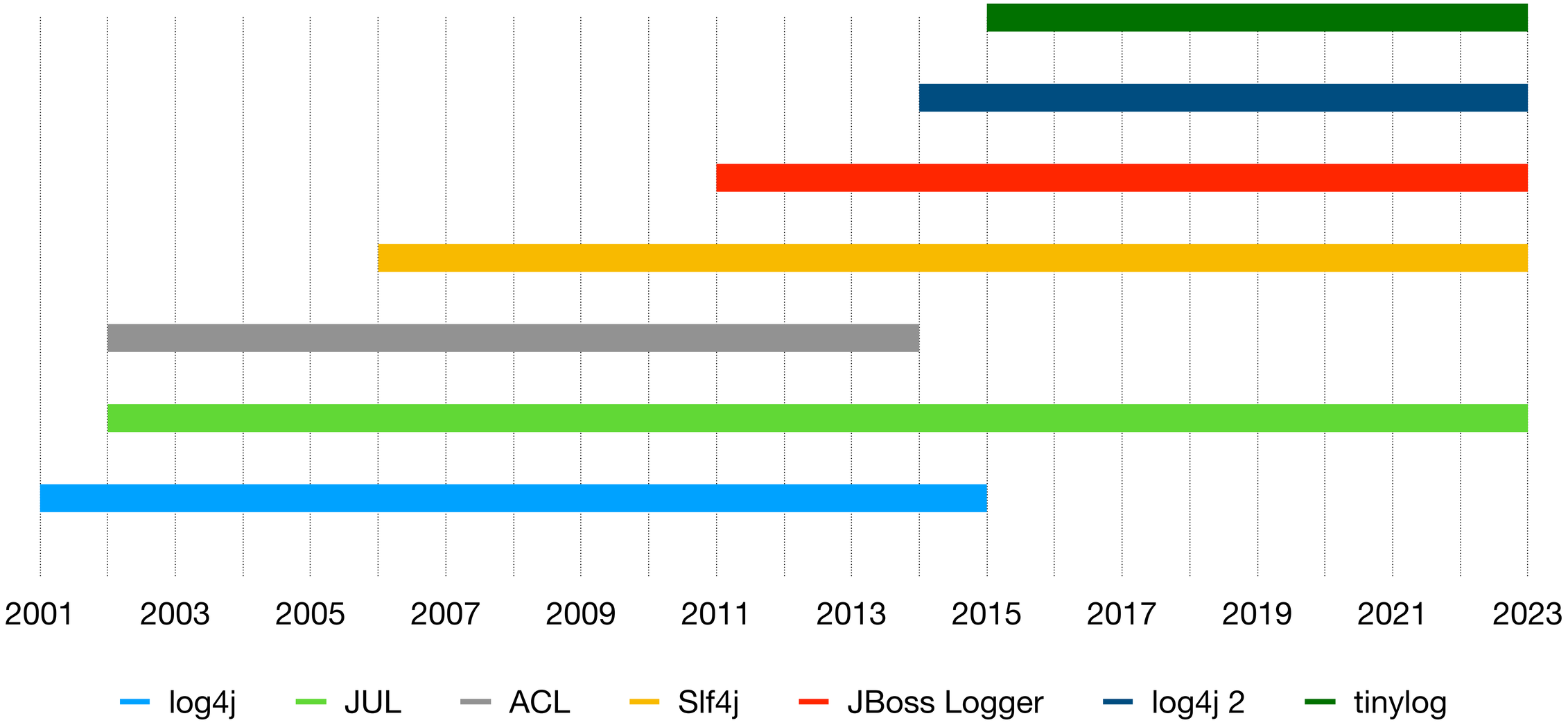
APIs für tinylog Backend

```
<dependency>
  <groupId>org.tinylog</groupId>
  <artifactId>log4j1.2-api</artifactId>
  <version>2.5.0</version>
</dependency>
<dependency>
  <groupId>org.tinylog</groupId>
  <artifactId>slf4j-tinylog</artifactId>
  <version>2.5.0</version>
</dependency>
<dependency>
  <groupId>org.tinylog</groupId>
  <artifactId>jboss-tinylog</artifactId>
  <version>2.5.0</version>
</dependency>
```

Fazit

Was man sich merken sollte

zeitlicher Überblick



Nur ein Backend verwenden

Nur ein Backend verwenden

- So gut wie alle Logausgaben lassen sich auf ein „Backend“ umbiegen.
- Damit ist nur noch eine Konfigurationsdatei notwendig.

In Bibliotheken nur API einbinden

In Bibliotheken nur API einbinden

- Es ist egal, welchen Logger eine Bibliothek einsetze.
- Diese sollte aber nur die API einbinden.

Fassaden werden nicht benötigt

Fassaden werden nicht benötigt

- Fassaden bringen keinen Mehrwert
  - Slf4j
  - Apache Commons Logging
  - JBoss Logging