

Einheit 05: Einführung in Log4J

L.Raed

Ludwig-Maximilians-Universität München
Institut für Informatik
Programmierung und Softwaretechnik
Prof. Wirsing

2. September 2013



Die Lernziele der heutigen Einheit

- 1 Grundlage von Log4J
- 2 Log4J in der Praxis einsetzen

Die Lernziele der heutigen Einheit

- 1 Grundlage von Log4J
- 2 Log4J in der Praxis einsetzen

Die Lernziele der heutigen Einheit

- 1 Grundlage von Log4J
- 2 Log4J in der Praxis einsetzen

Gliederung

1 Einführung in Log4J

- Prinzip und Installation
- Das Logger-Objekt
- Das Appender-Objekt
- Das Layout-Objekt
- Konfigurationsdatei

2 Log4J in der Praxis einsetzen

- Die drei Log4J Schritte
- Erstellung von log4j.properties
- Logger-Objekt im Code einsetzen

Prinzip und Installation

- Anforderungen an ein Logging Tool

- Mächtiger als die statische Console Print Statement.
- Fehler/Verhaltenmeldung in Text/Datenbank/E-mails direkt schreiben.
- Zur Laufzeit on the fly konfigurierbar ohne neu Kompilierung des Codes.

- Das Log4J Prinzip

- Schreibt ein Nachricht in ein Output formatiert und gemäß einer Priorität.
- Nachrichten-Priorität: debug, info, warnung, error, fatal
- Output-Ziel: Console, Text, Datei, Datenbank, E-mail.
- Java Logging basiert auf die Apache Log4J Tool.

- Log4J Installation

- 1 Log4J ist ein Logging Tool von Apache Software.
- 2 Log4J downloaden: <http://logging.apache.org/log4j/1.2/download.html>
- 3 Ein lib Ordner unter dem Projekt java_fortgeschrittene erzeugen.
- 4 Die log4J.jar Datei unter java_fortgeschrittene/lib speichern
- 5 Die log4J.jar Datei im Projekt-Classpath einbinden.

Prinzip und Installation

- Anforderungen an ein Logging Tool

- Mächtiger als die statische Console Print Statement.
- Fehler/Verhaltenmeldung in Text/Datenbank/E-mails direkt schreiben.
- Zur Laufzeit on the fly konfigurierbar ohne neu Kompilierung des Codes.

- Das Log4J Prinzip

- Schreibt eine Nachricht in ein Output formatiert und gemäß einer Priorität.
- Nachrichten-Priorität: debug, info, warnung, error, fatal
- Output-Ziel: Console, Text, Datei, Datenbank, E-mail.
- Java Logging basiert auf der Apache Log4J Tool.

- Log4J Installation

- 1 Log4J ist ein Logging Tool von Apache Software.
- 2 Log4J downloaden: <http://logging.apache.org/log4j/1.2/download.html>
- 3 Ein lib Ordner unter dem Projekt java_fortgeschrittene erzeugen.
- 4 Die log4J.jar Datei unter java_fortgeschrittene/lib speichern
- 5 Die log4J.jar Datei im Projekt-Classpath einbinden.

Prinzip und Installation

- Anforderungen an ein Logging Tool

- Mächtiger als die statische Console Print Statement.
- Fehler/Verhaltenmeldung in Text/Datenbank/E-mails direkt schreiben.
- Zur Laufzeit on the fly konfigurierbar ohne neu Kompilierung des Codes.

- Das Log4J Prinzip

- Schreibt eine Nachricht in ein Output formatiert und gemäß einer Priorität.
- Nachrichten-Priorität: debug, info, warnung, error, fatal
- Output-Ziel: Console, Text, Datei, Datenbank, E-mail.
- Java Logging basiert auf der Apache Log4J Tool.

- Log4J Installation

- 1 Log4J ist ein Logging Tool von Apache Software.
- 2 Log4J downloaden: <http://logging.apache.org/log4j/1.2/download.html>
- 3 Ein lib Ordner unter dem Projekt java_fortgeschrittene erzeugen.
- 4 Die log4J.jar Datei unter java_fortgeschrittene/lib speichern
- 5 Die log4J.jar Datei im Projekt-Classpath einbinden.

Prinzip und Installation

- Anforderungen an ein Logging Tool

- Mächtiger als die statische Console Print Statement.
- Fehler/Verhaltenmeldung in Text/Datenbank/E-mails direkt schreiben.
- Zur Laufzeit on the fly konfigurierbar ohne neu Kompilierung des Codes.

- Das Log4J Prinzip

- Schreibt ein Nachricht in ein Output formatiert und gemäß einer Priorität.
- Nachrichten-Priorität: debug, info, warnung, error, fatal
- Output-Ziel: Console, Text, Datei, Datenbank, E-mail.
- Java Logging basiert auf die Apache Log4J Tool.

- Log4J Installation

- 1 Log4J ist ein Logging Tool von Apache Software.
- 2 Log4J downloaden: <http://logging.apache.org/log4j/1.2/download.html>
- 3 Ein lib Ordner unter dem Projekt java_fortgeschrittene erzeugen.
- 4 Die log4J.jar Datei unter java_fortgeschrittene/lib speichern
- 5 Die log4J.jar Datei im Projekt-Classpath einbinden.

Prinzip und Installation

- Anforderungen an ein Logging Tool

- Mächtiger als die statische Console Print Statement.
- Fehler/Verhaltenmeldung in Text/Datenbank/E-mails direkt schreiben.
- Zur Laufzeit on the fly konfigurierbar ohne neu Kompilierung des Codes.

- Das Log4J Prinzip

- Schreibt ein Nachricht in ein Output formatiert und gemäß einer Priorität.
- Nachrichten-Priorität: debug, info, warnung, error, fatal
- Output-Ziel: Console, Text, Datei, Datenbank, E-mail.
- Java Logging basiert auf die Apache Log4J Tool.

- Log4J Installation

- 1 Log4J ist ein Logging Tool von Apache Software.
- 2 Log4J downloaden: <http://logging.apache.org/log4j/1.2/download.html>
- 3 Ein lib Ordner unter dem Projekt java_fortgeschrittene erzeugen.
- 4 Die log4J.jar Datei unter java_fortgeschrittene/lib speichern
- 5 Die log4J.jar Datei im Projekt-Classpath einbinden.

Prinzip und Installation

- Anforderungen an ein Logging Tool

- Mächtiger als die statische Console Print Statement.
- Fehler/Verhaltenmeldung in Text/Datenbank/E-mails direkt schreiben.
- Zur Laufzeit on the fly konfigurierbar ohne neu Kompilierung des Codes.

- Das Log4J Prinzip

- Schreibt ein Nachricht in ein Output formatiert und gemäß einer Priorität.
- Nachrichten-Priorität: debug, info, warnung, error, fatal
- Output-Ziel: Console, Text, Datei, Datenbank, E-mail.
- Java Logging basiert auf die Apache Log4J Tool.

- Log4J Installation

- 1 Log4J ist ein Logging Tool von Apache Software.
- 2 Log4J downloaden: <http://logging.apache.org/log4j/1.2/download.html>
- 3 Ein lib Ordner unter dem Projekt java_fortgeschrittene erzeugen.
- 4 Die log4J.jar Datei unter java_fortgeschrittene/lib speichern
- 5 Die log4J.jar Datei im Projekt-Classpath einbinden.

Wie funktioniert Log4J?

- Die nötigen Aktionen bei Log4J

- 1 Priorität festlegen: mit einem Logger Objekt
- 2 Output-Ziel festlegen: mit einem/mehreren Appender-Objekt(e)
- 3 Format festlegen: mit einem/mehreren Layout-Objekt(e)

- Funktionsweise: Logger ← Appender(s) ← Layout(s) Regel

- 1 Dem Logger Objekt wird ein/mehrere Appenders zugewiesen.
- 2 Jedem Appender-Objekt wird ein/mehrere Layout-Objekte zugewiesen.

Wie funktioniert Log4J?

- Die nötigen Aktionen bei Log4J

- 1 Priorität festlegen: mit einem Logger Objekt
- 2 Output-Ziel festlegen: mit einem/mehreren Appender-Objekt(e)
- 3 Format festlegen: mit einem/mehreren Layout-Objekt(e)

- Funktionsweise: Logger ← Appender(s) ← Layout(s) Regel

- 1 Dem Logger Objekt wird ein/mehrere Appenders zugewiesen.
- 2 Jedem Appender-Objekt wird ein/mehrere Layout-Objekte zugewiesen.

Wie funktioniert Log4J?

- Die nötigen Aktionen bei Log4J

- 1 Priorität festlegen: mit einem Logger Objekt
- 2 Output-Ziel festlegen: mit einem/mehreren Appender-Objekt(e)
- 3 Format festlegen: mit einem/mehreren Layout-Objekt(e)

- Funktionsweise: Logger ← Appender(s) ← Layout(s) Regel

- 1 Dem Logger Objekt wird ein/mehrere Appenders zugewiesen.
- 2 Jedem Appender-Objekt wird ein/mehrere Layout-Objekte zugewiesen.

Wie funktioniert Log4J?

- Die nötigen Aktionen bei Log4J

- 1 Priorität festlegen: mit einem Logger Objekt
- 2 Output-Ziel festlegen: mit einem/mehreren Appender-Objekt(e)
- 3 Format festlegen: mit einem/mehreren Layout-Objekt(e)

- Funktionsweise: Logger ← Appender(s) ← Layout(s) Regel

- 1 Dem Logger Objekt wird ein/mehrere Appenders zugewiesen.
- 2 Jedem Appender-Objekt wird ein/mehrere Layout-Objekte zugewiesen.

Anwendung und Einsatzgebiete von Log4J

- Log4J realisieren

- Hard-Kodiert: alle Zuweisungen werden programmiert.
- Konfiguration-File: enthält alle nötigen Zuweisungen und Regeln.
- Die Realisierung mit einer Konfigurationsdatei wird immer bevorzugt.

- Beispiele für die Einsatzgebiete von Log4J

- Senden von nützlichen Informationen für Debugging-Zwecke

```
logger.debug("Entering method");
logger.debug("Exiting method");
```
- Logging von IO-Exceptions.

```
catch(IOException e){logger.error("Cought: " + e);}
```
- E-mails Benachrichtigung bei Systemfehlern.

Anwendung und Einsatzgebiete von Log4J

- Log4J realisieren

- Hard-Kodiert: alle Zuweisungen werden programmiert.
- Konfiguration-File: enthält alle nötigen Zuweisungen und Regeln.
- Die Realisierung mit einer Konfigurationsdatei wird immer bevorzugt.

- Beispiele für die Einsatzgebiete von Log4J

- Senden von nützlichen Informationen für Debugging-Zwecke

```
logger.debug("Entering method");
logger.debug("Exiting method");
```
- Logging von IO-Exceptions.

```
catch(IOException e){logger.error("Caught: " + e);}
```
- E-mails Benachrichtigung bei Systemfehlern.

Anwendung und Einsatzgebiete von Log4J

- Log4J realisieren

- Hard-Kodiert: alle Zuweisungen werden programmiert.
- Konfiguration-File: enthält alle nötigen Zuweisungen und Regeln.
- Die Realisierung mit einer Konfigurationsdatei wird immer bevorzugt.

- Beispiele für die Einsatzgebiete von Log4J

- Senden von nützlichen Informationen für Debugging-Zwecke

```
logger.debug("Entering method");
logger.debug("Exiting method");
```
- Logging von IO-Exceptions.

```
catch(IOException e){logger.error("Caught: " + e);}
```
- E-mails Benachrichtigung bei Systemfehlern.

Anwendung und Einsatzgebiete von Log4J

- Log4J realisieren

- Hard-Kodiert: alle Zuweisungen werden programmiert.
- Konfiguration-File: enthält alle nötigen Zuweisungen und Regeln.
- Die Realisierung mit einer Konfigurationsdatei wird immer bevorzugt.

- Beispiele für die Einsatzgebiete von Log4J

- Senden von nützlichen Informationen für Debugging-Zwecke

```
logger.debug("Entering method");  
logger.debug("Exiting method");
```

- Logging von IO-Exceptions.

```
catch(IOException e){logger.error("Cought: " + e);
```

- E-mails Benachrichtigung bei Systemfehlern.

Das Logger-Objekt

- Was ist ein Logger-Objekt?
 - Ist ein Objekt zum Senden von Nachrichten.
 - Legt durch seine Methoden die Nachrichten-Priorität fest.
- Logger sind hierarchisch strukturiert
 - Der Wurzel-Logger wird RootLogger genannt.
 - Alle anderen (eigenen) Logger erben implizit von RootLogger.
 - `Logger.getRootLogger();` liefert den RootLogger zurück.
 - `Logger.getLogger();` definiert ein eignes Logger-Objekt.
- Es wird fast immer mit dem eigenen Logger gearbeitet

Das Logger-Objekt

- Was ist ein Logger-Objekt?
 - Ist ein Objekt zum Senden von Nachrichten.
 - Legt durch seine Methoden die Nachrichten-Priorität fest.
- Logger sind hierarchisch strukturiert
 - Der Wurzel-Logger wird RootLogger genannt.
 - Alle anderen (eigenen) Logger erben implizit von RootLogger.
 - `Logger.getRootLogger();` liefert den RootLogger zurück.
 - `Logger.getLogger();` definiert ein eignes Logger-Objekt.
- Es wird fast immer mit dem eigenen Logger gearbeitet

Das Logger-Objekt

- Was ist ein Logger-Objekt?
 - Ist ein Objekt zum Senden von Nachrichten.
 - Legt durch seine Methoden die Nachrichten-Priorität fest.
- Logger sind hierarchisch strukturiert
 - Der Wurzel-Logger wird RootLogger genannt.
 - Alle anderen (eigenen) Logger erben implizit von RootLogger.
 - `Logger.getRootLogger();` liefert den RootLogger zurück.
 - `Logger.getLogger();` definiert ein eignes Logger-Objekt.
- Es wird fast immer mit dem eigenen Logger gearbeitet

Das Logger-Objekt

- Was ist ein Logger-Objekt?
 - Ist ein Objekt zum Senden von Nachrichten.
 - Legt durch seine Methoden die Nachrichten-Priorität fest.
- Logger sind hierarchisch strukturiert
 - Der Wurzel-Logger wird RootLogger genannt.
 - Alle anderen (eigenen) Logger erben implizit von RootLogger.
 - `Logger.getRootLogger();` liefert den RootLogger zurück.
 - `Logger.getLogger();` definiert ein eignes Logger-Objekt.
- Es wird fast immer mit dem eigenen Logger gearbeitet

Das Logger-Objekt

- Was ist ein Logger-Objekt?
 - Ist ein Objekt zum Senden von Nachrichten.
 - Legt durch seine Methoden die Nachrichten-Priorität fest.
- Logger sind hierarchisch strukturiert
 - Der Wurzel-Logger wird RootLogger genannt.
 - Alle anderen (eigenen) Logger erben implizit von RootLogger.
 - `Logger.getRootLogger();` liefert den RootLogger zurück.
 - `Logger.getLogger();` definiert ein eignes Logger-Objekt.
- Es wird fast immer mit dem eigenen Logger gearbeitet

Das Logger-Objekt und die Priorität

- **Logger-Objekt Prioritäts-Methoden aufsteigend**

- 1 `debug(Object o)`: niedrigste Priorität für Debug-Zwecke
- 2 `info(Object o)`: für Informationen, Statistik.
- 3 `warn(Object o)`: für Warnmeldungen.
- 4 `error(Object o)`: für Fehler z.B. Exception
- 5 `fatal(Object o)` für höchste Priorität: schwerwiegende Fehler
Object o wird automatisch durch `toString()` in String konvertiert.

- **Welche Bedeutung und Sinn hat die Priorität?**

- bestimmt, ob eine Nachricht an das Output gesendet wird.
- ermöglicht verschiedene Information-Level.
- Beispiel: logge error aber keine User/Passwort Informationen.
- In Release: logge error/fatal in file und fatal auch in Console.
- In Test-Phase: logge einfach alles, auch user/passwort Information.

Das Logger-Objekt und die Priorität

- **Logger-Objekt Prioritäts-Methoden aufsteigend**

- 1 **debug(Object o)**: niedrigste Priorität für Debug-Zwecke
- 2 **info(Object o)**: für Informationen, Statistik.
- 3 **warn(Object o)**: für Warn meldungen.
- 4 **error(Object o)**: für Fehler z.B Exception
- 5 **fatal(Object o)** für höchste Priorität: schwerwiegende Fehler
Object o wird automatisch durch toString() in String konvertiert.

- **Welche Bedeutung und Sinn hat die Priorität?**

- bestimmt, ob eine Nachricht an das Output gesendet wird.
- ermöglicht verschiedene Information-Level.
- Beispiel: logge error aber keine User/Passwort Informationen.
- In Release: logge error/fatal in file und fatal auch in Console.
- In Test-Phase: logge einfach alles, auch user/passwort Information.

Das Logger-Objekt und die Priorität

- **Logger-Objekt Prioritäts-Methoden aufsteigend**

- 1 **debug(Object o)**: niedrigste Priorität für Debug-Zwecke
- 2 **info(Object o)**: für Informationen, Statistik.
- 3 **warn(Object o)**: für Warn meldungen.
- 4 **error(Object o)**: für Fehler z.B Exception
- 5 **fatal(Object o)** für höchste Priorität: schwerwiegende Fehler
Object o wird automatisch durch toString() in String konvertiert.

- **Welche Bedeutung und Sinn hat die Priorität?**

- bestimmt, ob eine Nachricht an das Output gesendet wird.
- ermöglicht verschiedene Information-Level.
- Beispiel: logge error aber keine User/Passwort Informationen.
- In Release: logge error/fatal in file und fatal auch in Console.
- In Test-Phase: logge einfach alles, auch user/passwort Information.

Das Logger-Objekt und die Priorität

- **Logger-Objekt Prioritäts-Methoden aufsteigend**

- 1 **debug(Object o)**: niedrigste Priorität für Debug-Zwecke
- 2 **info(Object o)**: für Informationen, Statistik.
- 3 **warn(Object o)**: für Warn meldungen.
- 4 **error(Object o)**: für Fehler z.B Exception
- 5 **fatal(Object o)** für höchste Priorität: schwerwiegende Fehler
Object o wird automatisch durch toString() in String konvertiert.

- **Welche Bedeutung und Sinn hat die Priorität?**

- bestimmt, ob eine Nachricht an das Output gesendet wird.
- ermöglicht verschiedene Information-Level.
- Beispiel: logge error aber keine User/Passwort Informationen.
- In Release: logge error/fatal in file und fatal auch in Console.
- In Test-Phase: logge einfach alles, auch user/passwort Information.

Das Appender-Objekt

- Ein Appender-Objekt

- bestimmt das Output-Ziel UND führt das Output durch.
- Ist also ein Objekt, das die Output durchführt.
- Das einfachste Appender ist der Console-Appender
- Der Console-Appender entspricht dem Console Print-Statement
- Appenders können in einer Datei, Datenbank und E-mails schreiben.

- Dem Logger-Objekt wird ein/mehrere Appenders zugewiesen

- Ein einziger Appender bedeutet ein einziges Output-Ziel.
- Mehrere Appenders bedeutet mehrere Output-Ziele (gleichzeitig).
- Beispiel: schreibe in log.txt, in der Console und in Datenbank.

Das Appender-Objekt

- Ein Appender-Objekt

- bestimmt das Output-Ziel UND führt das Output durch.
- Ist also ein Objekt, das die Output durchführt.
- Das einfachste Appender ist der Console-Appender
- Der Console-Appender entspricht dem Console Print-Statement
- Appenders können in einer Datei, Datenbank und E-mails schreiben.

- Dem Logger-Objekt wird ein/mehrere Appenders zugewiesen

- Ein einziger Appender bedeutet ein einziges Output-Ziel.
- Mehrere Appenders bedeutet mehrere Output-Ziele (gleichzeitig).
- Beispiel: schreibe in log.txt, in der Console und in Datenbank.

Das Appender-Objekt

- Ein Appender-Objekt

- bestimmt das Output-Ziel UND führt das Output durch.
- Ist also ein Objekt, das die Output durchführt.
- Das einfachste Appender ist der Console-Appender
- Der Console-Appender entspricht dem Console Print-Statement
- Appenders können in einer Datei, Datenbank und E-mails schreiben.

- Dem Logger-Objekt wird ein/mehrere Appenders zugewiesen

- Ein einziger Appender bedeutet ein einziges Output-Ziel.
- Mehrere Appenders bedeutet mehrere Output-Ziele (gleichzeitig).
- Beispiel: schreibe in log.txt, in der Console und in Datenbank.

Das Appender-Objekt

- Ein Appender-Objekt

- bestimmt das Output-Ziel UND führt das Output durch.
- Ist also ein Objekt, das die Output durchführt.
- Das einfachste Appender ist der Console-Appender
- Der Console-Appender entspricht dem Console Print-Statement
- Appenders können in einer Datei, Datenbank und E-mails schreiben.

- Dem Logger-Objekt wird ein/mehrere Appenders zugewiesen

- Ein einziger Appender bedeutet ein einziges Output-Ziel.
- Mehrere Appenders bedeutet mehrere Output-Ziele (gleichzeitig).
- Beispiel: schreibe in log.txt, in der Console und in Datenbank.

Welche Appenders gibt es?

- Die log4j Appender: Output durchführen

Appender	Beschreibung
Console Appender	Logge in der Console
File Appender	Logge in einer Datei
JDBCAppender	Logge in einer Datenbank
SMTPAppender	Logge in SMTP: Versende E-mails
SocketAppender	Logge in TCP Socket
NTEventLogAppender	Logge in Windows Event log
RollingFileAppenders	log.txt + Backup (ab file size x)
DailyRollingFileAppender	log.txt + Backup (ab time t)

- Für die Formatierung wird jedem Appender ein Layout zugewiesen

Welche Appenders gibt es?

- Die log4j Appender: Output durchführen

Appender	Beschreibung
Console Appender	Logge in der Console
File Appender	Logge in einer Datei
JDBCAppender	Logge in einer Datenbank
SMTPAppender	Logge in SMTP: Versende E-mails
SocketAppender	Logge in TCP Socket
NTEventLogAppender	Logge in Windows Event log
RollingFileAppenders	log.txt + Backup (ab file size x)
DailyRollingFileAppender	log.txt + Backup (ab time t)

- Für die Formatierung wird jedem Appender ein Layout zugewiesen

Welche Appenders gibt es?

- Die log4j Appender: Output durchführen

Appender	Beschreibung
Console Appender	Logge in der Console
File Appender	Logge in einer Datei
JDBCAppender	Logge in einer Datenbank
SMTPAppender	Logge in SMTP: Versende E-mails
SocketAppender	Logge in TCP Socket
NTEventLogAppender	Logge in Windows Event log
RollingFileAppenders	log.txt + Backup (ab file size x)
DailyRollingFileAppender	log.txt + Buckupt (ab time t)

- Für die Formatierung wird jedem Appender ein Layout zugewiesen

Das Layout-Objekt

- Ein Layout-Objekt

- Formatiert eine Nachricht gemäß eines „Format String“
- Format String = normaler Text Oder Conversion Specifier
- Normaler Text wird unverändert im Appender geschrieben.
- Conversion Specifier wird gemäss Formatierungsregel geschrieben.

- Formatierungsregel mit Conversion Specifier: %Zeichen

- Conversion Specifier beginnt mit einem % gefolgt von einem Zeichen.
- Das Zeichen nach % ist normalerweise ein Buchstabe.

Das Layout-Objekt

- Ein Layout-Objekt

- Formatiert eine Nachricht gemäß eines „Format String“
- Format String = normaler Text Oder Conversion Specifier
- Normaler Text wird unverändert im Appender geschrieben.
- Conversion Specifier wird gemäss Formatierungsregel geschrieben.

- Formatierungsregel mit Conversion Specifier: %Zeichen

- Conversion Specifier beginnt mit einem % gefolgt von einem Zeichen.
- Das Zeichen nach % ist normalerweise ein Buchstabe.

Das Layout-Objekt

- Ein Layout-Objekt

- Formatiert eine Nachricht gemäß eines „Format String“
- Format String = normaler Text Oder Conversion Specifier
- Normaler Text wird unverändert im Appender geschrieben.
- Conversion Specifier wird gemäss Formatierungsregel geschrieben.

- Formatierungsregel mit Conversion Specifier: %Zeichen

- Conversion Specifier beginnt mit einem % gefolgt von einem Zeichen.
- Das Zeichen nach % ist normalerweise ein Buchstabe.

Das Layout-Objekt

- Ein Layout-Objekt

- Formatiert eine Nachricht gemäß eines „Format String“
- Format String = normaler Text Oder Conversion Specifier
- Normaler Text wird unverändert im Appender geschrieben.
- Conversion Specifier wird gemäss Formatierungsregel geschrieben.

- Formatierungsregel mit Conversion Specifier: %Zeichen

- Conversion Specifier beginnt mit einem % gefolgt von einem Zeichen.
- Das Zeichen nach % ist normalerweise ein Buchstabe.

Log4J Conversion Specifiers

- Die log4j Conversion Specifiers

Specifier	Beschreibung
%C	Logger-Name
%d	Datum und Zeit, Default: ISO8601
%m	An den Logger übergebene Nachricht
%n	Neue Zeile (Platform-abhängig: n, r)
%p	Priorität der Nachricht
%r	Ablaufzeit in milliseconds.
%t	Thread-Name
%%	Prozent Zeichen
%C	Vollständiger qualifizierter Klassen-Name.
%F	Dateiname
%l	Location Information
%L	log.txt + Buckup (ab time t)

- %C,%F,%l,%L : geben nützliche Code-Infos, sind aber rechen-intensiv

Log4J Conversion Specifiers

- Die log4j Conversion Specifiers

Specifier	Beschreibung
<code>%c</code>	Logger-Name
<code>%d</code>	Datum und Zeit, Default: ISO8601
<code>%m</code>	An den Logger übergebene Nachricht
<code>%n</code>	Neue Zeile (Platform-abhängig: n, r)
<code>%p</code>	Priorität der Nachricht
<code>%r</code>	Ablaufzeit in milliseconds.
<code>%t</code>	Thread-Name
<code>%%</code>	Prozent Zeichen
<code>%C</code>	Vollständiger qualifizierter Klassen-Name.
<code>%F</code>	Dateiname
<code>%l</code>	Location Information
<code>%L</code>	log.txt + Buckup (ab time t)

- `%C,%F,%l,%L` : geben nützliche Code-Infos, sind aber rechen-intensiv

Log4J Conversion Specifiers

- Die log4j Conversion Specifiers

Specifier	Beschreibung
<code>%c</code>	Logger-Name
<code>%d</code>	Datum und Zeit, Default: ISO8601
<code>%m</code>	An den Logger übergebene Nachricht
<code>%n</code>	Neue Zeile (Platform-abhängig: n, r)
<code>%p</code>	Priorität der Nachricht
<code>%r</code>	Ablaufzeit in milliseconds.
<code>%t</code>	Thread-Name
<code>%%</code>	Prozent Zeichen
<code>%C</code>	Vollständiger qualifizierter Klassen-Name.
<code>%F</code>	Dateiname
<code>%l</code>	Location Information
<code>%L</code>	log.txt + Buckup (ab time t)

- `%C,%F,%l,%L` : geben nützliche Code-Infos, sind aber rechen-intensiv

Log4J Datum Formate

- Die log4j Datum Formate

log4j Format	SampleDateFormat	Beispiel
ABSOLUTE	hh:mm:ss,SSS	15:20:15,467
DATE	dd MMM YYYY hh:mm:ss,SSS	05 Feb 2010 ..
ISO8601	YYYY-mm-dd hh:mm:ss,SSS	2010-02-05 ..

- Beispiele: %d{log4j} und %d{SimpleDateFormat}

- %d{ABSOLUTE} ergibt z.B. 15:20:15,467
- %d{MMM d, YYYY hh:mm:ss a} ergibt z.B. Feb 8, 2010 15:20:15 pm

Log4J Datum Formate

- Die log4j Datum Formate

log4j Format	SampleDateFormat	Beispiel
ABSOLUTE	hh:mm:ss,SSS	15:20:15,467
DATE	dd MMM YYYY hh:mm:ss,SSS	05 Feb 2010 ..
ISO8601	YYYY-mm-dd hh:mm:ss,SSS	2010-02-05 ..

- Beispiele: %d{log4j} und %d{SimpleDateFormat}

- %d{ABSOLUTE} ergibt z.B. 15:20:15,467
- %d{MMM d, YYYY hh:mm:ss a} ergibt z.B. Feb 8, 2010 15:20:15 pm

Log4J Datum Formate

- Die log4j Datum Formate

log4j Format	SampleDateFormat	Beispiel
ABSOLUTE	hh:mm:ss,SSS	15:20:15,467
DATE	dd MMM YYYY hh:mm:ss,SSS	05 Feb 2010 ..
ISO8601	YYYY-mm-dd hh:mm:ss,SSS	2010-02-05 ..

- Beispiele: %d{log4j} und %d{SimpleDateFormat}

- %d{ABSOLUTE} ergibt z.B. 15:20:15,467
- %d{MMM d, YYYY hh:mm:ss a} ergibt z.B. Feb 8, 2010 15:20:15 pm

Log4J Datum Formate

- Die log4j Datum Formate

log4j Format	SampleDateFormat	Beispiel
ABSOLUTE	hh:mm:ss,SSS	15:20:15,467
DATE	dd MMM YYYY hh:mm:ss,SSS	05 Feb 2010 ..
ISO8601	YYYY-mm-dd hh:mm:ss,SSS	2010-02-05 ..

- Beispiele: %d{log4j} und %d{SimpleDateFormat}

- %d{ABSOLUTE} ergibt z.B. 15:20:15,467
- %d{MMM d, YYYY hh:mm:ss a} ergibt z.B. Feb 8, 2010 15:20:15 pm

Conversion Specifier optionale Formatierungszeichen

● Optionale Formatierungszeichen

- Sind optionale Specifier-Character zwischen dem % und dem Buchstabe.
- optionale - : Links ausrichten
- optionale n (Zahl): Minimale Breite (Versehen mit Spaces wenn nötig)
- optionale m: Maximale Breite (Daten von Links werden abgeschnitten)

● Optionale Formatierungsbeispiele

- Beschränke die Nachrichtlänge auf 50 Zeichen: %50m%n
- Liefere Dateiname und Zeile: (%F:%L)

Conversion Specifier optionale Formatierungszeichen

● Optionale Formatierungszeichen

- Sind optionale Specifier-Character zwischen dem % und dem Buchstabe.
- optionale - : Links ausrichten
- optionale n (Zahl): Minimale Breite (Versehen mit Spaces wenn nötig)
- optionale m: Maximale Breite (Daten von Links werden abgeschnitten)

● Optionale Formatierungsbeispiele

- Beschränke die Nachrichtlänge auf 50 Zeichen: %50m%n
- Liefere Dateiname und Zeile: (%F:%L)

Conversion Specifier optionale Formatierungszeichen

● Optionale Formatierungszeichen

- Sind optionale Specifier-Character zwischen dem % und dem Buchstabe.
- optionale - : Links ausrichten
- optionale n (Zahl): Minimale Breite (Versehen mit Spaces wenn nötig)
- optionale m: Maximale Breite (Daten von Links werden abgeschnitten)

● Optionale Formatierungsbeispiele

- Beschränke die Nachrichtlänge auf 50 Zeichen: %50m%n
- Liefere Dateiname und Zeile: (%F:%L)

Conversion Specifier optionale Formatierungszeichen

• Optionale Formatierungszeichen

- Sind optionale Specifier-Character zwischen dem % und dem Buchstabe.
- optionale - : Links ausrichten
- optionale n (Zahl): Minimale Breite (Versehen mit Spaces wenn nötig)
- optionale m: Maximale Breite (Daten von Links werden abgeschnitten)

• Optionale Formatierungsbeispiele

- Beschränke die Nachrichtlänge auf 50 Zeichen: %50m%n
- Liefere Dateiname und Zeile: (%F:%L)

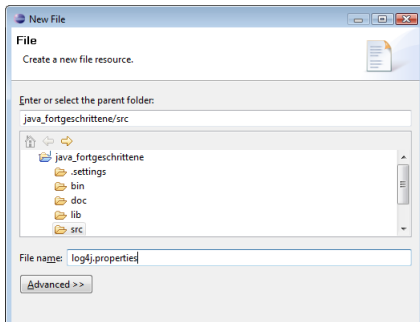
log4j.properties Konfigurationsdatei

● Arbeiten mit einer Konfigurationsdatei

- log4j sucht zuerst nach der Konfigurationsdatei log4j.properties.
- log4j.properties muss in den Ordner projektname/src gespeichert werden.
- Ist log4j.properties vorhanden, so benutzt log4j diese Konfigurationsdatei.
- In log4j.properties werden Logger, Appenders und Layouts definiert.

● Erstellen von log4j.properties

- src markieren → RM → New → File
- Enter or Select the Parent Folder: projektname/src übernehmen.
- File name: log4j.properties



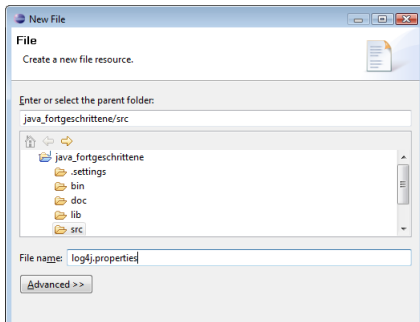
log4j.properties Konfigurationsdatei

• Arbeiten mit einer Konfigurationsdatei

- log4j sucht zuerst nach der Konfigurationsdatei log4j.properties.
- log4j.properties muss in den Ordner projektname/src gespeichert werden.
- Ist log4j.properties vorhanden, so benutzt log4j diese Konfigurationsdatei.
- In log4j.properties werden Logger, Appenders und Layouts definiert.

• Erstellen von log4j.properties

- src markieren → RM → New → File
- Enter or Select the Parent Folder: projektname/src übernehmen.
- File name: log4j.properties



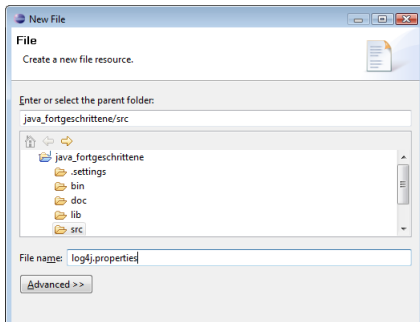
log4j.properties Konfigurationsdatei

• Arbeiten mit einer Konfigurationsdatei

- log4j sucht zuerst nach der Konfigurationsdatei log4j.properties.
- log4j.properties muss in den Ordner projektname/src gespeichert werden.
- Ist log4j.properties vorhanden, so benutzt log4j diese Konfigurationsdatei.
- In log4j.properties werden Logger, Appenders und Layouts definiert.

• Erstellen von log4j.properties

- src markieren → RM → New → File
- Enter or Select the Parent Folder: projektname/src übernehmen.
- File name: log4j.properties



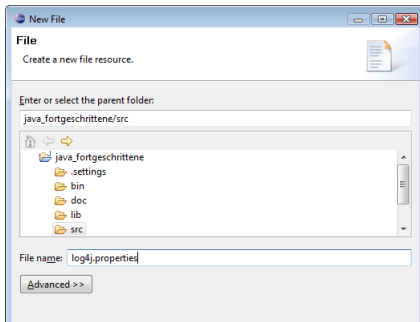
log4j.properties Konfigurationsdatei

• Arbeiten mit einer Konfigurationsdatei

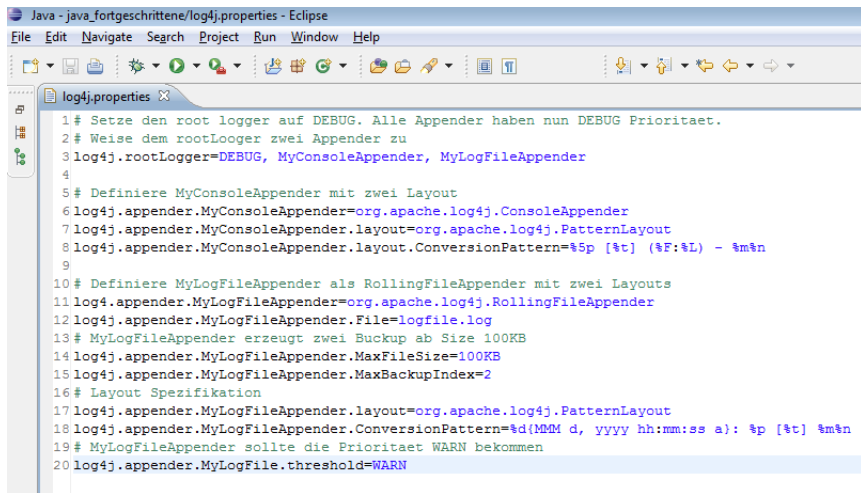
- log4j sucht zuerst nach der Konfigurationsdatei log4j.properties.
- log4j.properties muss in den Ordner projektname/src gespeichert werden.
- Ist log4j.properties vorhanden, so benutzt log4j diese Konfigurationsdatei.
- In log4j.properties werden Logger, Appenders und Layouts definiert.

• Erstellen von log4j.properties

- src markieren → RM → New → File
- Enter or Select the Parent Folder: projektname/src übernehmen.
- File name: log4j.properties



log4j.properties Beispiel



```
Java - java_fortgeschrittene/log4j.properties - Eclipse
File Edit Navigate Search Project Run Window Help

log4j.properties
1 # Setze den root logger auf DEBUG. Alle Appender haben nun DEBUG Prioritaet.
2 # Weise dem rootLogger zwei Appender zu
3 log4j.rootLogger=DEBUG, MyConsoleAppender, MyLogFileAppender
4
5 # Definiere MyConsoleAppender mit zwei Layout
6 log4j.appender.MyConsoleAppender=org.apache.log4j.ConsoleAppender
7 log4j.appender.MyConsoleAppender.layout=org.apache.log4j.PatternLayout
8 log4j.appender.MyConsoleAppender.layout.ConversionPattern=%5p [%t] (%F:%L) - %m%n
9
10 # Definiere MyLogFileAppender als RollingFileAppender mit zwei Layouts
11 log4j.appender.MyLogFileAppender=org.apache.log4j.RollingFileAppender
12 log4j.appender.MyLogFileAppender.File=logfile.log
13 # MyLogFileAppender erzeugt zwei Backup ab Size 100KB
14 log4j.appender.MyLogFileAppender.MaxFileSize=100KB
15 log4j.appender.MyLogFileAppender.MaxBackupIndex=2
16 # Layout Spezifikation
17 log4j.appender.MyLogFileAppender.layout=org.apache.log4j.PatternLayout
18 log4j.appender.MyLogFileAppender.layout.ConversionPattern=%d{MMM d, yyyy hh:mm:ss a}: %p [%t] %m%n
19 # MyLogFileAppender sollte die Prioritaet WARN bekommen
20 log4j.appender.MyLogFileAppender.threshold=WARN
```

Die drei Log4J Schritte

1 Erstelle eine log4.properties Datei

- Spezifiziere die Prioritätslevel des RootLogger Objekts.
- Spezifiziere die Appenders, die dem Logger-Objekt zugewiesen werden.
- Spezifiziere die Layouts, die jedem Appender zugewiesen werden.

2 Logging Code für jede Klasse schreiben

- *static Logger logger = Logger.getLogger(klassenname.class);*
- Erzeugt im Programm ein eigenes Logger-Objekt
- logger erbt implizit von RootLogger
- logger hat also alles, was in log4j.properties steht.

3 Logger-Objekt im Code einsetzen

- Um Debug Informationen zu loggen.
- Um IO Exceptions zu loggen.
- Um Warnungen und Info Meldungen zu loggen.
- Um wichtige Ereignisse/Aktionen/Methoden zu loggen.

Die drei Log4J Schritte

1 Erstelle eine log4.properties Datei

- Spezifiziere die Prioritätslevel des RootLogger Objekts.
- Spezifiziere die Appenders, die dem Logger-Objekt zugewiesen werden.
- Spezifiziere die Layouts, die jedem Appender zugewiesen werden.

2 Logging Code für jede Klasse schreiben

- *static Logger logger = Logger.getLogger(klassenname.class);*
- Erzeugt im Programm ein eigenes Logger-Objekt
- logger erbt implizit von RootLogger
- logger hat also alles, was in log4j.properties steht.

3 Logger-Objekt im Code einsetzen

- Um Debug Informationen zu loggen.
- Um IO Exceptions zu loggen.
- Um Warnungen und Info Meldungen zu loggen.
- Um wichtige Ereignisse/Aktionen/Methoden zu loggen.

Die drei Log4J Schritte

1 Erstelle eine log4.properties Datei

- Spezifiziere die Prioritätslevel des RootLogger Objekts.
- Spezifiziere die Appenders, die dem Logger-Objekt zugewiesen werden.
- Spezifiziere die Layouts, die jedem Appender zugewiesen werden.

2 Logging Code für jede Klasse schreiben

- *static Logger logger = Logger.getLogger(klassenname.class);*
- Erzeugt im Programm ein eigenes Logger-Objekt
- logger erbt implizit von RootLogger
- logger hat also alles, was in log4j.properties steht.

3 Logger-Objekt im Code einsetzen

- Um Debug Informationen zu loggen.
- Um IO Exceptions zu loggen.
- Um Warnungen und Info Meldungen zu loggen.
- Um wichtige Ereignisse/Aktionen/Methoden zu loggen.

Die drei Log4J Schritte

1 Erstellte eine log4.properties Datei

- Spezifiziere die Prioritätslevel des RootLogger Objekts.
- Spezifiziere die Appenders, die dem Logger-Objekt zugewiesen werden.
- Spezifiziere die Layouts, die jedem Appender zugewiesen werden.

2 Logging Code für jede Klasse schreiben

- *static Logger logger = Logger.getLogger(klassenname.class);*
- Erzeugt im Programm ein eigenes Logger-Objekt
- logger erbt implizit von RootLogger
- logger hat also alles, was in log4j.properties steht.

3 Logger-Objekt im Code einsetzen

- Um Debug Informationen zu loggen.
- Um IO Exceptions zu loggen.
- Um Warnungen und Info Meldungen zu loggen.
- Um wichtige Ereignisse/Aktionen/Methoden zu loggen.

Die drei Log4J Schritte

1 Erstelle eine log4.properties Datei

- Spezifiziere die Prioritätslevel des RootLogger Objekts.
- Spezifiziere die Appenders, die dem Logger-Objekt zugewiesen werden.
- Spezifiziere die Layouts, die jedem Appender zugewiesen werden.

2 Logging Code für jede Klasse schreiben

- *static Logger logger = Logger.getLogger(klassenname.class);*
- Erzeugt im Programm ein eigenes Logger-Objekt
- logger erbt implizit von RootLogger
- logger hat also alles, was in log4j.properties steht.

3 Logger-Objekt im Code einsetzen

- Um Debug Informationen zu loggen.
- Um IO Exceptions zu loggen.
- Um Warnungen und Info Meldungen zu loggen.
- Um wichtige Ereignisse/Aktionen/Methoden zu loggen.

Die drei Log4J Schritte

1 Erstellte eine log4.properties Datei

- Spezifiziere die Prioritätslevel des RootLogger Objekts.
- Spezifiziere die Appenders, die dem Logger-Objekt zugewiesen werden.
- Spezifiziere die Layouts, die jedem Appender zugewiesen werden.

2 Logging Code für jede Klasse schreiben

- *static Logger logger = Logger.getLogger(klassenname.class);*
- Erzeugt im Programm ein eigenes Logger-Objekt
- logger erbt implizit von RootLogger
- logger hat also alles, was in log4j.properties steht.

3 Logger-Objekt im Code einsetzen

- Um Debug Informationen zu loggen.
- Um IO Exceptions zu loggen.
- Um Warnungen und Info Meldungen zu loggen.
- Um wichtige Ereignisse/Aktionen/Methoden zu loggen.

Syntax von log4j.properties

1 Weise dem RootLogger einer Priorität und Appenders zu

- `log4j.rootLogger= PriorityLevel, Appender1,..., AppenderX`
- `PriorityLevel` = DEBUG, INFO, WARN, ERROR, FATAL
- Logge alle Nachrichten mit Priorität \geq `PriorityLevel`
- `PriorityLevel=DEBUG` \rightarrow Logge alle Nachrichten.

2 Definiere die Appenders und setze ihre Priority ggf. ihre Properties

- `log4j.appender.AppenderX= org.apache.log4j.AppenderClass`
- `log4j.appender.AppenderX.Property=Value`
- `log4j.appender.AppenderX.threshold=Neue PriorityLevel`

3 Weise jedem Appender ein/mehrere Layout zu

- `log4j.appender.AppenderX.layout=org.apache.log4j.PatternLayout`
- `log4j.appender.AppenderX.layout.ConversionPattern=Specifiers`

Syntax von log4j.properties

1 Weise dem RootLogger einer Priorität und Appenders zu

- `log4j.rootLogger= PriorityLevel, Appender1,..., AppenderX`
- `PriorityLevel` = DEBUG, INFO, WARN, ERROR, FATAL
- Logge alle Nachrichten mit Priorität \geq `PriorityLevel`
- `PriorityLevel=DEBUG` \rightarrow Logge alle Nachrichten.

2 Definiere die Appenders und setze ihre Priority ggf. ihre Properties

- `log4j.appender.AppenderX= org.apache.log4j.AppenderClass`
- `log4j.appender.AppenderX.Property=Value`
- `log4j.appender.AppenderX.threshold=Neue PriorityLevel`

3 Weise jedem Appender ein/mehrere Layout zu

- `log4j.appender.AppenderX.layout=org.apache.log4j.PatternLayout`
- `log4j.appender.AppenderX.layout.ConversionPattern=Specifiers`

Syntax von log4j.properties

1 Weise dem RootLogger einer Priorität und Appenders zu

- `log4j.rootLogger= PriorityLevel, Appender1,..., AppenderX`
- `PriorityLevel` = DEBUG, INFO, WARN, ERROR, FATAL
- Logge alle Nachrichten mit Priorität \geq `PriorityLevel`
- `PriorityLevel=DEBUG` \rightarrow Logge alle Nachrichten.

2 Definiere die Appenders und setze ihre Priority ggf. ihre Properties

- `log4j.appender.AppenderX= org.apache.log4j.AppenderClass`
- `log4j.appender.AppenderX.Property=Value`
- `log4j.appender.AppenderX.threshold=Neue PriorityLevel`

3 Weise jedem Appender ein/mehrere Layout zu

- `log4j.appender.AppenderX.layout=org.apache.log4j.PatternLayout`
- `log4j.appender.AppenderX.layout.ConversionPattern=Specifiers`

Syntax von log4j.properties

- ❶ Weise dem RootLogger einer Priorität und Appenders zu
 - `log4j.rootLogger= PriorityLevel, Appender1,..., AppenderX`
 - `PriorityLevel` = DEBUG, INFO, WARN, ERROR, FATAL
 - Logge alle Nachrichten mit Priorität \geq `PriorityLevel`
 - `PriorityLevel=DEBUG` \rightarrow Logge alle Nachrichten.
- ❷ Definiere die Appenders und setze ihre Priority ggf. ihre Properties
 - `log4j.appender.AppenderX= org.apache.log4j.AppenderClass`
 - `log4j.appender.AppenderX.Property=Value`
 - `log4j.appender.AppenderX.threshold=Neue PriorityLevel`
- ❸ Weise jedem Appender ein/mehrere Layout zu
 - `log4j.appender.AppenderX.layout=org.apache.log4j.PatternLayout`
 - `log4j.appender.AppenderX.layout.ConversionPattern=Specifiers`

Syntax von log4j.properties

- ① Weise dem RootLogger einer Priorität und Appenders zu
 - `log4j.rootLogger= PriorityLevel, Appender1,..., AppenderX`
 - `PriorityLevel = DEBUG, INFO, WARN, ERROR, FATAL`
 - Logge alle Nachrichten mit Priorität \geq `PriorityLevel`
 - `PriorityLevel=DEBUG` → Logge alle Nachrichten.
- ② Definiere die Appenders und setze ihre Priority ggf. ihre Properties
 - `log4j.appender.AppenderX= org.apache.log4j.AppenderClass`
 - `log4j.appender.AppenderX.Property=Value`
 - `log4j.appender.AppenderX.threshold=Neue PriorityLevel`
- ③ Weise jedem Appender ein/mehrere Layout zu
 - `log4j.appender.AppenderX.layout=org.apache.log4j.PatternLayout`
 - `log4j.appender.AppenderX.layout.ConversionPattern=Specifiers`

Syntax von log4j.properties

- ① Weise dem RootLogger einer Priorität und Appenders zu
 - `log4j.rootLogger= PriorityLevel, Appender1,..., AppenderX`
 - `PriorityLevel` = DEBUG, INFO, WARN, ERROR, FATAL
 - Logge alle Nachrichten mit Priorität \geq `PriorityLevel`
 - `PriorityLevel=DEBUG` \rightarrow Logge alle Nachrichten.
- ② Definiere die Appenders und setze ihre Priority ggf. ihre Properties
 - `log4j.appender.AppenderX= org.apache.log4j.AppenderClass`
 - `log4j.appender.AppenderX.Property=Value`
 - `log4j.appender.AppenderX.threshold=Neue PriorityLevel`
- ③ Weise jedem Appender ein/mehrere Layout zu
 - `log4j.appenderAppenderX.layout=org.apache.log4j.PatternLayout`
 - `log4j.appender.AppenderX.layout.ConversionPattern=Specifiers`

Syntax von log4j.properties

- ① Weise dem RootLogger einer Priorität und Appenders zu
 - `log4j.rootLogger= PriorityLevel, Appender1,..., AppenderX`
 - `PriorityLevel` = DEBUG, INFO, WARN, ERROR, FATAL
 - Logge alle Nachrichten mit Priorität \geq `PriorityLevel`
 - `PriorityLevel=DEBUG` \rightarrow Logge alle Nachrichten.
- ② Definiere die Appenders und setze ihre Priority ggf. ihre Properties
 - `log4j.appender.AppenderX= org.apache.log4j.AppenderClass`
 - `log4j.appender.AppenderX.Property=Value`
 - `log4j.appender.AppenderX.threshold=Neue PriorityLevel`
- ③ Weise jedem Appender ein/mehrere Layout zu
 - `log4j.appenderAppenderX.layout=org.apache.log4j.PatternLayout`
 - `log4j.appender.AppenderX.layout.ConversionPattern=Specifiers`

Syntax von log4j.properties

- ① Weise dem RootLogger einer Priorität und Appenders zu
 - `log4j.rootLogger= PriorityLevel, Appender1,..., AppenderX`
 - `PriorityLevel` = DEBUG, INFO, WARN, ERROR, FATAL
 - Logge alle Nachrichten mit Priorität \geq `PriorityLevel`
 - `PriorityLevel=DEBUG` \rightarrow Logge alle Nachrichten.
- ② Definiere die Appenders und setze ihre Priority ggf. ihre Properties
 - `log4j.appender.AppenderX= org.apache.log4j.AppenderClass`
 - `log4j.appender.AppenderX.Property=Value`
 - `log4j.appender.AppenderX.threshold=Neue PriorityLevel`
- ③ Weise jedem Appender ein/mehrere Layout zu
 - `log4j.appender.AppenderX.layout=org.apache.log4j.PatternLayout`
 - `log4j.appender.AppenderX.layout.ConversionPattern=Specifiers`

Syntax von log4j.properties

- ① Weise dem RootLogger einer Priorität und Appenders zu
 - `log4j.rootLogger= PriorityLevel, Appender1,..., AppenderX`
 - `PriorityLevel` = DEBUG, INFO, WARN, ERROR, FATAL
 - Logge alle Nachrichten mit Priorität \geq `PriorityLevel`
 - `PriorityLevel=DEBUG` \rightarrow Logge alle Nachrichten.
- ② Definiere die Appenders und setze ihre Priority ggf. ihre Properties
 - `log4j.appender.AppenderX= org.apache.log4j.AppenderClass`
 - `log4j.appender.AppenderX.Property=Value`
 - `log4j.appender.AppenderX.threshold=Neue PriorityLevel`
- ③ Weise jedem Appender ein/mehrere Layout zu
 - `log4j.appenderAppenderX.layout=org.apache.log4j.PatternLayout`
 - `log4j.appender.AppenderX.layout.ConversionPattern=Specifiers`

Syntax von log4j.properties

- ① Weise dem RootLogger einer Priorität und Appenders zu
 - `log4j.rootLogger= PriorityLevel, Appender1,..., AppenderX`
 - `PriorityLevel` = DEBUG, INFO, WARN, ERROR, FATAL
 - Logge alle Nachrichten mit Priorität \geq `PriorityLevel`
 - `PriorityLevel=DEBUG` \rightarrow Logge alle Nachrichten.
- ② Definiere die Appenders und setze ihre Priority ggf. ihre Properties
 - `log4j.appender.AppenderX= org.apache.log4j.AppenderClass`
 - `log4j.appender.AppenderX.Property=Value`
 - `log4j.appender.AppenderX.threshold=Neue PriorityLevel`
- ③ Weise jedem Appender ein/mehrere Layout zu
 - `log4j.appenderAppenderX.layout=org.apache.log4j.PatternLayout`
 - `log4j.appender.AppenderX.layout.ConversionPattern=Specifiers`

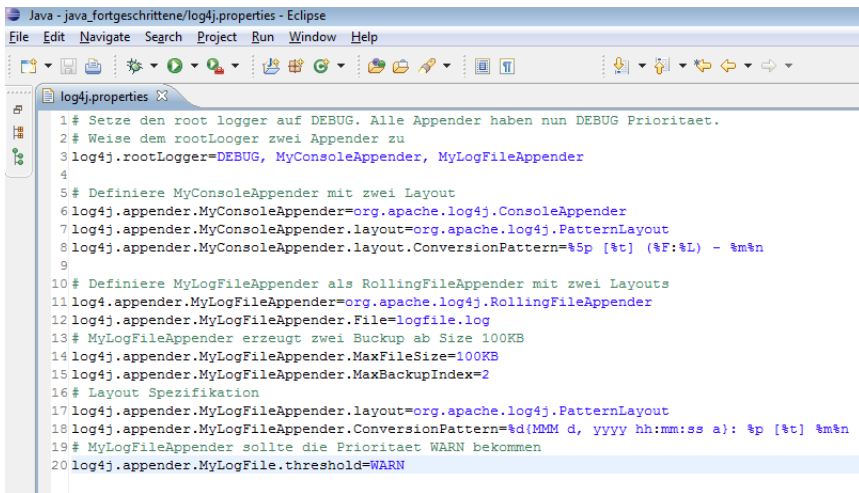
Syntax von log4j.properties

- ① Weise dem RootLogger einer Priorität und Appenders zu
 - `log4j.rootLogger= PriorityLevel, Appender1,..., AppenderX`
 - `PriorityLevel` = DEBUG, INFO, WARN, ERROR, FATAL
 - Logge alle Nachrichten mit Priorität \geq `PriorityLevel`
 - `PriorityLevel=DEBUG` \rightarrow Logge alle Nachrichten.
- ② Definiere die Appenders und setze ihre Priority ggf. ihre Properties
 - `log4j.appender.AppenderX= org.apache.log4j.AppenderClass`
 - `log4j.appender.AppenderX.Property=Value`
 - `log4j.appender.AppenderX.threshold=Neue PriorityLevel`
- ③ Weise jedem Appender ein/mehrere Layout zu
 - `log4j.appenderAppenderX.layout=org.apache.log4j.PatternLayout`
 - `log4j.appender.AppenderX.layout.ConversionPattern=Specifiers`

Syntax von log4j.properties

- ① Weise dem RootLogger einer Priorität und Appenders zu
 - `log4j.rootLogger= PriorityLevel, Appender1,..., AppenderX`
 - `PriorityLevel` = DEBUG, INFO, WARN, ERROR, FATAL
 - Logge alle Nachrichten mit Priorität \geq `PriorityLevel`
 - `PriorityLevel=DEBUG` \rightarrow Logge alle Nachrichten.
- ② Definiere die Appenders und setze ihre Priority ggf. ihre Properties
 - `log4j.appender.AppenderX= org.apache.log4j.AppenderClass`
 - `log4j.appender.AppenderX.Property=Value`
 - `log4j.appender.AppenderX.threshold=Neue PriorityLevel`
- ③ Weise jedem Appender ein/mehrere Layout zu
 - `log4j.appenderAppenderX.layout=org.apache.log4j.PatternLayout`
 - `log4j.appender.AppenderX.layout.ConversionPattern=Specifiers`

log4j.properties Beispiel



```
1 # Setze den root logger auf DEBUG. Alle Appender haben nun DEBUG Prioritaet.
2 # Weise dem rootLogger zwei Appender zu
3 log4j.rootLogger=DEBUG, MyConsoleAppender, MyLogFileAppender
4
5 # Definiere MyConsoleAppender mit zwei Layout
6 log4j.appender.MyConsoleAppender=org.apache.log4j.ConsoleAppender
7 log4j.appender.MyConsoleAppender.layout=org.apache.log4j.PatternLayout
8 log4j.appender.MyConsoleAppender.layout.ConversionPattern=%5p [%t] (%F:%L) - %m%n
9
10 # Definiere MyLogFileAppender als RollingFileAppender mit zwei Layouts
11 log4j.appender.MyLogFileAppender=org.apache.log4j.RollingFileAppender
12 log4j.appender.MyLogFileAppender.File=logfile.log
13 # MyLogFileAppender erzeugt zwei Backup ab Size 100KB
14 log4j.appender.MyLogFileAppender.MaxFileSize=100KB
15 log4j.appender.MyLogFileAppender.MaxBackupIndex=2
16 # Layout Spezifikation
17 log4j.appender.MyLogFileAppender.layout=org.apache.log4j.PatternLayout
18 log4j.appender.MyLogFileAppender.layout.ConversionPattern=%d{MMM d, yyyy hh:mm:ss a}: %p [%t] %m%n
19 # MyLogFileAppender sollte die Prioritaet WARN bekommen
20 log4j.appender.MyLogFileAppender.threshold=WARN
```

log4j im Code benutzen

1 log4j.properties erstellen

- RootLogger Priorität und Appenders definieren.
- Appenders mit ihren Layout und Properties definieren.
- RootLogger Priorität für Appenders bei Bedarf anpassen.

2 Logging Code für jede Klasse schreiben

- `static Logger logger = Logger.getLogger(klassenname.class);`

3 Gewünschte Codeabschnitte loggen

- `methodeX(){logger.debug("Entering methodeX()")}`
- `if(x>y){..} else{logger.warn("y muss kleiner als x sein");}`
- `if(gefunden){..} else{logger.warn("kann key " + key + " nicht finden");}`

Beispiel für log4j.properties

1 Erzeugen Sie für Factorial Beispiel ein log4j.properties

- Binden Sie zuerst die neuste log4j JAR in Ihrem Projekt ein (Einheit 1).
- Appenders sind die Console und ein log File
- Formatierung: PatternLayout und ConversionPattern
- Erzeugen Sie Ihr eigenes Logger-Objekt im Code.
- Testen Sie Ihre Konfigurationsdatei log4j.properties.

2 Erweitern Sie Ihr Programm, so daß

- die Zahl, deren Fakultät berechnet wird, wird von der Console eingelesen.
- danach fordert Sie das Programm auf, eine neue Zahl anzugeben.
- Nur wenn Sie dann -1 eingeben, hört das Programm auf.
- Jetzt ändern Sie (während das Programm läuft) die log4j.properties.
- Testen Sie on the fly Ihre Änderungen.

• **TIPP 1: Sehen Sie die log4j.properties Syntax und Beispiel Folien**

• **TIPP II: <http://logging.apache.org/log4j/1.2/manual.html>**