

Logging

Carsten Gips (FH Bielefeld)

Unless otherwise noted, this work is licensed under CC BY-SA 4.0.

Wie prüfen Sie die Werte von Variablen/Objekten?

1. Debugging

- Beeinflusst Code nicht
- Kann schnell komplex und umständlich werden
- Sitzung transient – nicht wiederholbar

2. “Poor-man’s-debugging” (Ausgaben mit `System.out.println`)

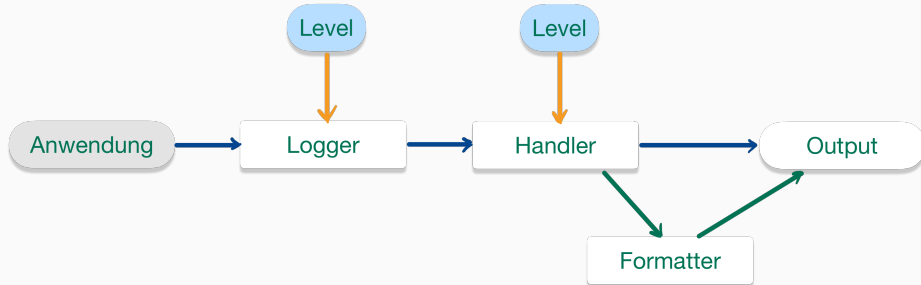
- Müssen irgendwann entfernt werden
- Ausgabe nur auf einem Kanal (Konsole)
- Keine Filterung nach Problemgrad – keine Unterscheidung zwischen Warnungen, einfachen Informationen, ...

3. Logging

- Verschiedene (Java-) Frameworks:
`java.util.logging` (JDK), *log4j* (Apache), *SLF4J*, *Logback*, ...

Java Logging API – Überblick

Paket `java.util.logging`



Konsole: `logging.LoggingDemo`

Erzeugen neuer Logger

```
import java.util.logging.Logger;  
Logger l = Logger.getLogger(MyClass.class.getName());
```

- **Factory-Methode** der Klasse `java.util.logging.Logger`

```
public static Logger getLogger(String name);
```

=> Methode liefert bereits **vorhandenen Logger** mit diesem Namen

- **Best Practice:**

Nutzung des voll-qualifizierten Klassennamen: `MyClass.class.getName()`

- Leicht zu implementieren
- Leicht zu erklären
- Spiegelt modulares Design
- Ausgaben enthalten automatisch Hinweis auf Herkunft (Lokalität) der Meldung
- **Alternativen:** Funktionale Namen wie "XML", "DB", "Security"

Ausgabe von Logmeldungen

```
public void log(Level level, String msg);
```

- Diverse Convenience-Methoden (Auswahl):

```
public void warning(String msg)
public void info(String msg)
public void entering(String srcClass, String srcMethod)
public void exiting(String srcClass, String srcMethod)
```

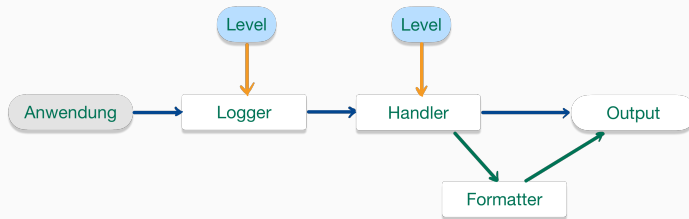
- Beispiel

```
import java.util.logging.Logger;
Logger l = Logger.getLogger(MyClass.class.getName());
l.info("Hello World :-)");
```

Wichtigkeit von Logmeldungen: Stufen

- `java.util.logging.Level` definiert 7 Stufen:
 - `SEVERE`, `WARNING`, `INFO`, `CONFIG`, `FINE`, `FINER`, `FINEST`
(von höchster zu niedrigster Prio)
 - Zusätzlich `ALL` und `OFF`
- Nutzung der Log-Level:
 - Logger hat Log-Level: Meldungen mit kleinerem Level werden verworfen
 - Prüfung mit `public boolean isLoggable(Level)`
 - Setzen mit `public void setLevel(Level)`

Jemand muss die Arbeit machen ...



- Pro Logger **mehrere** Handler möglich
 - Logger übergibt nicht verworfene Nachrichten an Handler
 - Handler haben selbst ein Log-Level (analog zum Logger)
 - Handler verarbeiten die Nachrichten, wenn Level ausreichend
- Standard-Handler: `StreamHandler`, `ConsoleHandler`, `FileHandler`
- Handler nutzen zur Formatierung der Ausgabe einen `Formatter`
- Standard-Formatter: `SimpleFormatter` und `XMLFormatter`

- Logger bilden **Hierarchie** über Namen
 - Trenner für Namenshierarchie: "." (analog zu Packages)
 - Jeder Logger kennt seinen Eltern-Logger: `Logger#getParent()`
 - Basis-Logger: leerer Name ("")
 - Voreingestelltes Level des Basis-Loggers: `Level.INFO` (!)
- Weiterleiten von Nachrichten
 - Nicht verworfene Log-Aufrufe werden an Eltern-Logger weitergeleitet
 - Abschalten mit `Logger#setUseParentHandlers(false);`
 - Diese leiten an ihren Eltern-Logger weiter (unabhängig von Log-Level!)

- Java Logging API im Paket `java.util.logging`
- Neuer Logger über **Factory-Methode** der Klasse `Logger`
 - Einstellbares Log-Level (Klasse `Level`)
 - Handler kümmern sich um die Ausgabe, nutzen dazu Formatter
 - Mehrere Handler je Logger registrierbar
 - Log-Level auch für Handler einstellbar (!)
 - Logger (und Handler) “interessieren” sich nur für Meldungen ab bestimmter Wichtigkeit
 - Logger reichen nicht verworfene Meldungen defaultmäßig an Eltern-Logger weiter (rekursiv)



Unless otherwise noted, this work is licensed under CC BY-SA 4.0.