Wie Sie wirklich brauchbare Ablaufprotokolle erstellen

Die 11 goldenen Logging-Regeln

Das Protokollieren des Programmablaufs („Logging“) ist eines der wirkungsvollsten Werkzeuge zur Fehlersuche. Glücklicherweise existieren für Java-Entwickler leistungsfähige und dabei einfach anzuwendende Logging-Bibliotheken, wie etwa log4j oder die Java Logging API.

Also die Ärmel hochgekrempelt, flugs ein paar Jar-Dateien in den Klassenpfad versenkt, Pakete importiert und schon ist das Thema „Logging“ erledigt...

...aber: Protokollieren alleine hilft nicht weiter – die Meldungen müssen auch aussagekräftig sein! Wie formulieren Sie also wirklich brauchbare Ablaufprotokolle? Die folgenden 11 Regeln geben Antwort:

1 *Alle Meldungen in Englisch abfassen.*
2 *Jede Meldung als vollständigen Satz formulieren.*
3 *Jede Meldung freistehend formulieren.*
4 *Eine Meldung nicht der Formatierung wegen auf mehrere Zeilen aufteilen.*
5 *Textvariablen immer in einfache Anführungszeichen setzen.*
6 *Meldungen so formulieren, daß sie im Code verbleiben können.*
7 *Meldungen immer mit einem Punkt abschließen.*
8 *Meldungen, die eine Exception ausgeben, mit Doppelpunkt abschließen.*
9 *Wird der Beginn eines Zeitraumes angezeigt, die Meldung mit drei Punkten (...) beenden.*
10 *Wird das Ende eines Zeitraumes angezeigt, die Meldung mit einem Punkt (.) beenden.*
11 *Besser den Beginn als das Ende einer längeren Operation anzeigen.*

An Linie ausschneiden und auf die Rückseite Ihres Nokia Communicators kleben.

Die Regeln zu kennen und zu befolgen ist die eine Sache. Eine andere ist, die Gründe dahinter zu verstehen. Im Folgenden wird daher jede Regel mit ihrer Motivation sowie einem schlechten Beispiel (~~`durchgestrichen`~~) und einer passenden Verbesserung (`nicht durchgestrichen`) vorgestellt.

1 Alle Meldungen in Englisch abfassen.

Englische Texte sind im Vergleich zu deutschen in der Regel knapper und dazu international einsetzbar. Da Fachbegriffe meist auf Englisch eingeführt werden, wird so zudem ein Deutsch-Englisch-Kauderwelsch verhindert.

```
~~log.debug("Benötigten Satz saumäßig dringender Aktualisierungen gedownloadet.");~~
log.debug("Downloaded set of hot fixes.");
```

2 Jede Meldung als vollständigen Satz formulieren.

Faustregel: Die Meldung enthält mindestens ein Objekt und ein Verb. Grund: Beim Entwickeln neuer Funktionen ist die Bedeutung einer Logging-Meldung zwar glasklar – bei der Fehlersuche ein paar Monate später jedoch längst nicht mehr in jedem Fall.

```
~~log.debug("User / Password");~~
log.debug("User deleted. Password removed from database.");
```

3 Jede Meldung freistehend formulieren.

„Freistehend“ bedeutet, daß eine Meldung für sich genommen eine eigene Aussage bildet und nicht nur im Kontext von mehreren umgebenden Meldungen verständlich ist.

Dadurch bleibt die Aussagekraft einer Meldung auch beim teilweisen Ausblenden der umgebenden Meldungen erhalten (z.B. durch Neukonfiguration der Verbosity-Levels).

Zusätzlich machen Sie die Meldungen unabhängiger von ihrer Reihenfolge zueinander. Dies kann während eines Refactorings äußerst praktisch sein, weil diese inhaltlichen Abhängigkeiten nicht automatisiert angepasst werden könnten.

Zugegeben, manchmal kann die Forderung nach Eigenständigkeit lange und ausführliche Meldungen erzeugen. Vergessen Sie aber nicht, daß Sie auf der Pirsch nach Fehlern immer problemlos Ihre Protokolle mit Filtern destillieren, aber nicht eine einzige ausführlichere Meldung nachträglich einfügen können.

```
log.debug("Heating up warp core...");              // Variante A
log.error("... Way too hot!");
...
log.error("... Shutting down...");

log.debug("Heating up warp core...");              // Variante B
...
log.error("Warp core overheated. Shutting down...");
```

Beispiel: Im Produktionssystem haben Sie die Ausführlichkeit der Protokollierung auf `WARN` gestellt. Welche Meldungen würden Sie im Falle einer kritischen Überhitzung des Warpkerns bevorzugen?

```
[ERROR] ... Way too hot!                           // Variante A: Wer? SevenOfNine?
[ERROR] ... Shutting down...                       // Was wird heruntergefahren?

[ERROR] Warp core overheated. Shutting down...     // Variante B
```

4 Eine Meldung nicht der Formatierung wegen auf mehrere Zeilen aufteilen.

Immer das „Eine Zeile – Eine Meldung“-Schema einhalten. Protokolldateien können enorme Umfänge erreichen. Gerade dann ist es nicht unüblich, mit zeilenbasierten Filterwerkzeugen wie etwa den Gebrüdern `grep`, `sed` und `awk` nach bestimmten Meldungen zu suchen. Ein Aufteilen einer Meldung auf mehrere Zeilen erschwert die automatische Verarbeitung jedoch enorm.

```
log.error("Warp core error:\n...Overheated due to:\n...'" + cause + "'!");
log.error("Warp core overheated due to: '" + cause + "'.");
```

5 Textvariablen immer in einfache Anführungszeichen setzen.

Dadurch werden Leerstrings bzw. vorausgehende oder nachfolgende Leerzeichen sichtbar.

```
log.warn("Cannot delete user " + username + ".");     // Variante A
log.warn("Cannot delete user '" + username + "'.");  // Variante B
```

Beispiel: Durch einen Programmierfehler ist die Variable `username` mit einem Nullstring initialisiert. Die obigen Logging-Befehle erzeugen dann folgende Ausgaben:

```
[WARN ] Cannot delete user .                       // Variante A
[WARN ] Cannot delete user ''.                     // Variante B
```

Bei welcher Variante hätten Sie die Natur des Fehlers zuerst erkannt?

6 Meldungen so formulieren, daß sie im Code verbleiben können.

Der Gigabit-Switch glüht, das Server-RAID knirscht und das Release-Datum ist bereits zwei Wochen überschritten. Gute Meldungestexte ausdenken? Keine Zeit! Ein `xxx` beim Betreten, ein `ciao` beim Verlassen der Funktion reicht – wird ja sowieso gleich wieder entfernt. Und pro Programmstart ein `##################` in die Protokolldatei – das macht die Darstellung im Konsolenfenster doch so schön übersichtlich...

So zugespitzt sich dies lesen mag, genau so oft lässt sich diese Art des Loggings in der freien Wildbahn beobachten – gerne noch gesteigert durch Verzicht auf Logging-APIs und die ausschließliche Verwendung von `System.out.println()` Anweisungen.

Formulieren Sie Ihre Meldungen besser so, daß sie im Code verbleiben können. Dies erfordert zwar ein intensives Ringen um die treffende Formulierung, doch es zahlt sich aus, denn:

- Die Adhoc-Meldungen müssten Sie später ja wieder entfernen (Mehraufwand!).
- Die entfernten Meldungen hätten ein paar Wochen später vielleicht den entscheidenden Hinweis bei einer Fehlersuche liefern können.
- Übersehene, im Code verbliebene kryptische Adhoc-Meldungen irritieren Sie später.
- Was auf Ihrer Konsole übersichtlich aussieht, kann bei anderen Entwicklern einfach nur sperrig sein. Benutzen Sie besser einen intelligenten Logfile-Betrachter. Das Eclipse-Plugin LogWatcher (`graysky.sourceforge.net`) etwa kann anhand von Signalwörtern einzelne Zeilen farblich hervorheben.

Bonuspunkt: Wer aussagekräftige Meldungen schreibt, hat sein Programm auch wirklich verstanden.

```
~~log.info("################");~~
~~log.debug("xxx");~~
~~...~~
~~log.debug("ciao");~~

log.info("Warp drive control " + APP_VERSION + " starting...");
log.debug("Calculating warp sequence...");
...
log.debug("Calculated warp sequence.");
```

7 Meldungen immer mit einem Punkt abschließen.

Dadurch können Sie die Vollständigkeit einer Logging-Meldung im Protokoll überprüfen. Oft werden Protokolle aus Geschwindigkeitsgründen mit gepufferten Dateien realisiert. Das bedeutet, daß die Ausgaben jeweils nur dann in die Protokolldatei geschrieben werden, wenn z.B. 4 Kilobyte an Daten zusammengekommen sind. Dadurch kann die neuste Logging-Meldung nur unvollständig im Protokoll sichtbar sein.

```
~~log.warn("Shutdown of warp core aborted");   // Variante A~~
log.warn("Shutdown of warp core aborted.");  // Variante B
```

Beispiel: Die untenstehenden Ausgaben zeigen jeweils die letzte Zeile einer Protokolldatei. Können Sie mit Sicherheit sagen, ob der Warpkern gerade heruntergefahren wird?

```
[WARN ] Shutdown of warp core                // Variante A: Unklar.
[WARN ] Shutdown of warp core                // Variante B: Hier fehlt doch was!
```

8 Meldungen, die eine Exception ausgeben, mit Doppelpunkt abschließen.

Damit weisen Sie darauf hin, daß eine Exception-Meldung in der Protokolldatei folgen muß. Fehlt diese, ist die Datei unvollständig.

```
~~log.warn("File could not be opened.");~~
log.warn("File could not be opened: ", ioe);
```

9 Wird der Beginn eines Zeitraumes angezeigt, die Meldung mit drei Punkten (...) beenden.

Dadurch weisen Sie darauf hin, daß nun ein längerer Programmabschnitt bearbeitet wird, der sich ggf. über mehrere nachfolgende Logging-Meldungen erstreckt. Das Verb in der Meldung ist immer in der Present Continuous Form (die sog. „Ing-Form“) dekliniert.

```
~~log.info("Start HTTP server");~~
log.info("Starting HTTP server...");
```

10 Wird das Ende eines Zeitraumes angezeigt, die Meldung mit einem Punkt (.) beenden.

Dadurch weisen Sie darauf hin, daß nun ein längerer Programmabschnitt beendet wird, der zusammen mit einer vorangegangenen Logging-Meldung eine Klammer bilden kann. Wählen Sie in diesem Fall die Formulierung sehr ähnlich, um die Zusammengehörigkeit zu betonen. Das Verb in der Meldung ist immer in der Simple Past Form dekliniert.

Alternativ: Sie wiederholten die Meldung zu Beginn des Zeitraumes und stellen ein `Done` voran.

```
~~log.info("HTTP Server o.k.");~~
log.info("HTTP Server started.");         // Alternative 1
log.info("Done starting HTTP Server.");   // Alternative 2
```

11 Besser den Beginn als das Ende einer längeren Operation anzeigen.

Im Zweifel `Starting HTTP server...` statt `HTTP Server started.` ausgeben. Warum?

Stürzt Ihnen das Programm beim Starten des Servers ab, würden Sie die letztere Meldung nämlich nie angezeigt bekommen. Das Protokoll kann nur festhalten, wo an welcher Stelle das Programm schon vorbeigekommen ist, aber nicht, wo es nie hinkommen wird.