# Nachdenkzettel: Logging

1. a: Kennzeichnen Sie in der nachfolgenden Konfiguration die Stellen wo entschieden wird,
   - was geloggt wird
   - wieviel geloggt wird
   - wo geloggt wird
   - wie geloggt wird

```
<Configuration>
    <Appenders>
        <File name="A1" fileName="A1.log" append="false">
            <PatternLayout pattern="%t %-5p %c{2} - %m%n"/>
        </File>
        <Console name="STDOUT" target="SYSTEM_OUT">
            <PatternLayout pattern="%d %-5p [%t] %C{2} (%F:%L) -
%m%n"/>
        </Console>
    </Appenders>

    <Loggers>
        <Logger name="se2examples.core.businessLogic.VehicleManager"
level="debug">
            <AppenderRef ref="A1"/>
        </Logger>
        <Root level="debug">
            <AppenderRef ref="STDOUT"/>
        </Root>
    </Loggers>
</Configuration>
```

   b: Könnte man auch alle Klassen eines Packages in eine eigene Datei loggen lassen? Wie?

2. Geben Sie ein Beispiel wann Sie die folgenden Log-Level verwenden würden:
   - `error`
   - `info`
   - `debug`

3. Macht ein Logging-Framework Ihre Anwendung langsamer? Wie sollte sich ein Logging-Framework idealerweise verhalten?

# Nachdenkzettel

-Logging-

## Aufgabe 1) b.

Ja das ist möglich, indem man einen Appender definiert und diesen dann an einen Logger zweist, der den Namen des packages trägt, beispielsweise: „se2examples.core.businessLogic" (sofern es sich bei businesslogic um ein package handelt)

## Aufgabe 2)

- error: Wenn Fehler auftreten, beispielsweise beim werfen einer Exception
- info: Allgemeine Informationen, wie beispielsweise „Spiel gestartet"
- debug: Alle Informationen die fürs Debugging wichtig werden könnten, wie Positionen eines Spielers auf dem Spielfeld.

## Aufgabe 3)

Es kann die Anwendung verlangsamen, beispielsweise dann, wenn sehr viele Logs in Dateien geschrieben werden. Dies sind I/O-operationen, die mehr Ressourcen verbrauchen als normale Speicheroperationen.
Idealerweise verhält sich das Logging-Framework so, dass es möglichst ressourcenschonend arbeitet.