**Write code (preferably test automation example) to implement various logging concepts.**

<https://github.com/AvietLoboTest/Logging-Concepts>

Logback and SLF4J: -

Using [Logback](https://logback.qos.ch/) in combination with [SLF4J](https://www.slf4j.org/). Let’s add two Maven dependencies to get started:

<properties>

<logback.version>1.3.0-alpha4</logback.version>

<slf4j.version>2.0.0-alpha1</slf4j.version>

</properties>

<dependencies>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>${slf4j.version}</version>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>${logback.version}</version>

</dependency>

</dependencies>

[Logback is configured](http://logback.qos.ch/manual/configuration.html) using a file called logback.xml (or logback-test.xml, check the details in the docs!) on the classpath. To start, we use the following configuration which defines a log appender that will print the results of all loggers to the console:

<?xml version="1.0"?>

<configuration>

<appender name="STDOUT"

class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>%cyan(%d{HH:mm:ss.SSS}) %highlight(%.-1level) %highlight(%msg) %n

</pattern>

</encoder>

</appender>

<root level="info">

<appender-ref ref="STDOUT" />

</root>

<logger name="qa.justtestlah" level="info" />

</configuration>

This example already makes use of ANSI colouring (for example, %cyan(...)). This can be helpful when reading logs during local development (on a console which supports ANSI colours) but will, obviously, not work when saving the log to plain text files.

Also, note that we include the timestamp %d{HH:mm:ss.SSS} in each logline. This seems like a no-brainer, but we have seen logs without it and the frustration it causes in reproducing bugs.

## **How to write to the log?**

The easiest way to create logger instances is using LoggerFactory like this:

private static final Logger LOG = LoggerFactory.getLogger("loggerName");

LOG.info("performing action " + actionName);

Another common practice is to use the class in which a logger is defined as its name:

private static final Logger LOG = LoggerFactory.getLogger(SomeJavaClass.class);

## **Parameterized logging**

We do not have to use String concatenation to build log messages. SLF4J supports a useful feature called [parameterized logging](http://logback.qos.ch/manual/architecture.html#parametrized):

LOG.info("Logging in user {} with password xxx on instance {}", user.getUsername(), instance.getURL());

To organize large amounts of log information, we can utilize different [log levels](https://dzone.com/articles/logging-levels-what-they-are-and-how-they-help-you)

It supports TRACE, DEBUG, INFO, WARN and ERROR

# Test logs for automated UI tests

In regards to UI automation, these are the actions we suggest having in the logs for each test execution:

* Version information of the application under test
* Information about the test device/test browser (OS version, screen size, device model etc.)
* Start and end of each test scenario (timestamps in sync with the machine(s) running the application under test)
* Start and end of each test step (timestamps in sync with the machine(s) running the application under test)
* Every interaction with the UI, i.e. clicking a button, entering text, scrolling, swiping etc.)
* Every background call to the system under test or a test double (for example, an API call to prepare a testing state). This should include the request and response (for example, response code on INFO and the full output on DEBUG level)
* Every interaction with the test framework (setting up a WebDriver, starting a browser instance, installing an app etc.)
* Every verification and its expected and actual results