Coding Assignment

Introduction & Background

Thank you for showing interest in our Java Engineer role within Developer Tools & Services at Credit Suisse! In order to be an engineer working on the Credit Suisse build, release & deployment platform, a strong understanding of Java programming is essential to being a successful member of the team. We understand that asking candidates to complete this challenge prior to any interviews may be strange - we do this because the criticality of this technical skill is essential to our goal to build a team of strong engineers who can work to build, grow, and maintain the bank's strategic tool chain.

All coding challenge submissions are reviewed by senior technical engineers on our team, and give us a good idea for your coding style. We ask all candidates across the globe to complete this challenge as we feel it aids in ensuring that only highly skilled and competent java engineers move forward in our recruitment process.

The challenge below is a generic java programming task that should take no more than 1-2 hours to complete. If you find it begins to take you significantly longer than 2 hours, don't worry - simply submit what you have completed and provide a few bullets of what your next steps would be.

The challenge can be found directly below - please be sure to submit your final product to Github, along with instructions on how to run / test the program from the command line.

Requirements for this coding assignment:

- Java 8
- Use of any open-source library is allowed
- Your program must use a gradle build system to resolve dependencies, build and test

Summary of task

Our custom-build server logs different events to a file. Every event has **2** entries in a log - one entry when the event was started and another when the event was finished. The entries in a log file have no specific order (it can occur that a specific event is logged before the event starts)

Every line in the file is a JSON object containing event data:

- **id** - the unique event identifier
- **state** - whether the event was started or finished (can have values "STARTED" or "FINISHED"
- **timestamp** - the timestamp of the event in milliseconds

Application Server logs also have the additional attributes:

- **type** - type of log
- **host** - hostname

Example:

```
{"id":"scsmbstgra", "state":"STARTED", "type":"APPLICATION_LOG",
"host":"12345", "timestamp":1491377495212}
{"id":"scsmbstgrb", "state":"STARTED", "timestamp":1491377495213}
{"id":"scsmbstgrc", "state":"FINISHED", "timestamp":1491377495218}
{"id":"scsmbstgra", "state":"FINISHED", "type":"APPLICATION_LOG",
"host":"12345", "timestamp":1491377495217}
{"id":"scsmbstgrc", "state":"STARTED", "timestamp":1491377495210}
{"id":"scsmbstgrb", "state":"FINISHED", "timestamp":1491377495216}
...
```

In the example above, the event **scsmbstgrb** duration is 1401377495216 - 1491377495213 = 3ms

The longest event is **scsmbstgrc** (1491377495218 - 1491377495210 = 8ms)

The program should:

- Take the input file path as input argument
- Flag any long events that take longer than 4ms with a column in the database called "alert"
- Write the found event details to file-based HSQLDB (http://hsqldb.org/) in the working folder
    - The application should a new table if necessary and enter the following values:
        - Event id
        - Event duration
        - Type and Host if applicable
        - "alert" true is applicable

Additional points will be granted for:

- Proper use of info and debug logging
- Proper use of Object Oriented programming
- Unit test coverage
- Multi-threaded solution
- Program that can handle very large files (gigabytes)

As stated above, submissions should be loaded onto Github.

Initial gradle file example

build.gradle

```
group 'com.test'
version '1.0-SNAPSHOT'

apply plugin: 'java'

sourceCompatibility = 1.7

repositories {
    mavenCentral()
}

dependencies {
    compile group: 'ch.qos.logback', name: 'logback-classic', version:
'1.1.7'

    testCompile group: 'junit', name: 'junit', version: '4.12'
}
```