Victoria University of Wellington

SWEN 301 Structured Methods

Assignment 2 (Individual) 2021

Objectives

The aim of this assignment is to (1) extend a real-world software system, and (2) to select, assess and use an existing open source component. This requires understanding the system to be extended, including its architecture, design and code organisation. The system is log4j version 1.2.17. The assignment also will expose you to runtime monitoring, a widely used industry practice.

Contribution to Final Grade

25 %

Managing Code

You are expected to manage your source code. It is required to use an **ECS GitLab repository** for this assignment, it is recommended to use the pre-created repository **https://gitlab.ecs.vuw.ac.nz/course-work/swen301/2021/<username>/Assignment2** .
We expect that the repository is actually used, i.e. you should frequently commit. One final commit at the end is not acceptable and will be penalised.

It is expected that you use the exact names stipulated in the assignment brief. This is a requirement that will be strictly enforced (you will lose points for violations as acceptance tests will fail), in particular as the importance of naming conventions is part of the discussion of this course. So for instance, in task 3, the brief requires a class **nz.ac.wgtn.swen301.assignment2.JSONLayoutTest**. This means: use exactly this package name and exactly this (local) class name (e.g., **JsonLayoutTest** and **JSONLayoutTests** are both wrong, and the package name has to match exactly -- note that there is no leading test in the package name, it is possible to have the same package in **src/main/java** and src/test/java).

What and How to submit

You must tag the submission, and submit a text file via the ECS submission system (link on course wiki) containing the URL of the tag, this will look something like this:

`https://gitlab.ecs.vuw.ac.nz/course-work/swen301/2021/Tom/assignment2/-/tags/submitted`

To mark the project, we will clone the repository and checkout the respective tag. Repositories have been created you can use, following this naming pattern:

`https://gitlab.ecs.vuw.ac.nz/course-work/swen301/2021/<username>/Assignment2`

To be marked, the project must be a valid Maven project, i.e. at least the following command must succeed when executed in the project root folder on a standard ECS lab machine:

`mvn compile`

Other maven commands (like **`mvn test`** and **`mvn package`**) will be used during marking as well to assess particular aspects of the assignment.

Make sure that sources and **`pom.xml`** are all **included**, but files used and generated by IDEs (IntelliJ, Eclipse) to manage project settings and files generated by Maven during the build are **excluded**. Submit only the minimum required. There are no requirements w.r.t. IDEs / tools that can be used, all mainstream Java IDEs all have Maven support built-in or supported via plugins. A submission link will be provided on the course wiki.

Prepare

Work through a “getting started” - type tutorials on log4j on the web, such as https://www.mkyong.com/logging/log4j-hello-world-example/ . **Note that the version we use is 1.2.17**. To better understand the design of log4j, read the following article: https://www.javaworld.com/article/2078767/java-tip-orthogonality-by-example.html .

In particular, you need to understand the concepts **layout** and **appender** before you start. A log4j lab will be (has been) offered in week 6 to prepare you for this assignment.

Background

The system is log4j, its purpose and architecture will be (or have been) discussed in the lectures, for additional resources check the **Prepare** section. The tasks focus around writing a memory appender. An appender is “where the logs go” -- examples are the console (System.out) or the file appender. Logging usually incurs some significant performance overhead as appenders perform I/O operations. In this assignment, your task is to design and implement an appender that by default does not use I/O , but stores logs in memory. There is still a function to export logs from the memory store on demand.

Tasks and Marking Schedule

Work **individually** to create the following program in Java version 8, using the Maven project structure and layout.

1. Create a maven project with the group name **nz.ac.wgtn.swen301.assignment2** and the artifact name **assignment2-<your studentid>**. Enforce the Java 8 requirement (for both compilation source and target) in the Maven using the **<properties>** tag (i.e., in the POM). **[2 marks]**
2. In this project, create a *log4j layout* **nz.ac.wgtn.swen301.assignment2.JSONLayout**, this layout can convert log events (i.e., instances of **org.apache.log4j.spi.LoggingEvent**) to JSON strings. I.e., the strings produced are valid JSON objects. You must use one of the following libraries to assist with generating JSON, and also parsing JSON (this is necessary for testing, task 3): json.org , gson, jackson, fastjson, jsonp. Please check the example in the appendix for the format of the JSON to be produced. In particular, only the attributes listed in the example must be included. **[3 marks]**
3. Write junit tests in a class **nz.ac.wgtn.swen301.assignment2.JSONLayoutTest** to test **JSONLayout**. You must include tests which **parse** the JSON produced, and comparing the parsed data with the attributes of the original log events used for testing. See Task 2 for a list of libraries that can be used to parse JSON. Aim for high coverage. Use annotation-based junit4 or junit5 tests, junit3-style tests based on making patterns and subclassing are not acceptable. **[3 marks]**
4. In this project, create a log4j appender **nz.ac.wgtn.swen301.assignment2.MemAppender** as a Java class**:**
    a. **MemAppender** stores all log events (i.e., instances of **org.apache.log4j.spi.LoggingEvent**) in a list (i.e., instance of **java.util.List**). **[1 mark]**
    b. **MemAppender** have a **name** property[1] of type **String**. **[1 mark]**
    c. Logs can be accessed using the following non-static method: **java.util.List<LoggingEvent> getCurrentLogs()** **[1 mark]**
    d. the list returned by **getCurrentLogs()** must not be modifiable **[1 mark]**
    e. **MemAppender** has a **maxSize** property of type **long** -- this is the maximum number of logs it can store. If the number of logs reaches **maxSize**, the oldest logs are discarded so that they can be garbage-collected, in order to avoid memory leaks (this is to be interpreted as inclusive, i.e. a state where there are actually **maxSize** logs is acceptable). The number of discarded logs is counted, and this count can be accessed using the **getDiscardedLogCount()** method returning a **long**. Set the default value of maxSize 1000. **[2 marks]**
    f. **MemAppender** has a method to export stored log events to a JSON file: **void exportToJSON(String fileName)**. This method exports the log events currently stored into a JSON file. The JSON file should contain a JSON array

[1] Note that a Java property refers to a pair of matching getters and setters method, in most cases associated with a private instance variable.

containing JSON objects. Each JSON object represents a log event, the log event properties are the keys in this JSON object. An example of a JSON file produced is provided below. The function must use the **JSONLayout** described earlier. **[2 marks]**

5. Write junit tests in a class **nz.ac.wgtn.swen301.assignment2.MemAppenderTest** that test **MemAppender** in combination with different loggers and levels. Aim for high coverage. Use annotation-based junit4 or junit5 tests, junit3-style tests based on making patterns and subclassing are not acceptable. **[3 marks]**
6. Create an MBean object for each instance of the **MemAppender** to add JMX monitoring to this object. The name of the mbean must include the value of the name property of the appender **[2 marks]**.
   The properties to be monitored are:
   a. The log events converted to strings as array : **String[] getLogs()** , the string representation of a LoggingEvent is obtained using **org.apache.log4j.PatternLayout** with the default conversion pattern.
   b. The number of logs: **long getLogCount()**
   c. The number of discarded logs: **long getDiscardedLogCount()**
   d. An action to export the log events in the memory appender to a file in JSON format (i.e., as an array of json objects): **void exportToJSON(String fileName)**
7. Implement a class **nz.ac.wgtn.swen301.assignment2.example.LogRunner t**hat is executable (i.e., has a main method). When executed, it will run for 2 mins, and produce one log event per second. The log level and message of this event is to be randomised, the main purpose of this class is to test the mbean. **[1 mark]**
8. Create a report in **README.md** in markdown format that contains the following sections:
   a. Any special Instructions if needed
   b. A discussion why you chose a particular JSON library. Base your decision on your experience (if any), documentation, technical aspects (e.g. performance as shown in the stress tests, stability, number and size of direct and indirect dependencies), and social aspects (size and activity of developer community, license, support like mailing lists and stackoverflow topics, usage by others, …) **[3 marks]**

Penalties (up to -2 for each kind of violation)

1. violations of naming rules
2. violating the Maven standard project layout
3. use of absolute references (e.g., libraries should not be referenced using absolute paths like “C:\\Users\\..”, instead use relative references w.r.t. the project root folder)
4. references to local libraries (libraries should be referenced via the Maven repository)
5. use of libraries which are not whitelisted in task 2 (unless they are only used via transitive dependencies)
6. not using git for development, or only using a very few large commits at the end of

the project

7. committing binaries or IDE meta data to the repository (such as eclipse .project or .classpath, IntelliJ .idea , Maven target/ folders etc

Appendices

Testing MBeans

To monitor mbeans at runtime, you can use **jvisualvm**. You need to install the MBean plugin from Tools > Plugins. Then you can see your MBean when the application runs, query its properties and execute its action(s). As you need a long-running execution to do this, it is recommend to create an executable class LogRunner that runs for an extended period of time (ca 1 min) and keeps on producing logs which are appended to the memory appender.

Do not make this a test (@Test annotated) as this will mean that mvn test will time out or run forever !

JSON Export Format

This is an example of the JSON format to be produced by the export function -- only the following 5 attributes logger, level, starttime, thread and message need to be represented, note that all property names are lower-case -- this is part of the requirements !

```
[
   {
      "logger":"foo",
      "level":"WARN",
      "starttime":"1584016200",
      "thread":"main",
      "message":"something went wrong"
   },
   {
      "logger":"foo",
      "level":"DEBUG",
      "starttime":"1584016222",
      "thread":"main",
      "message":"interesting !"
   }
]
```

Change History

Date	Change
21/04/21	Corrected repo URL to use pre-created repos.
21/04/21	Clarified that the output of JSONLayout is described in the example in the appendix.
26/04/21	Clarified 4e that **maxSize** can be interpreted as inclusive
28/04/21	In task 7, changed “one log event per minute” to “one log event per second”