

Victoria University of Wellington

SWEN 301 Structured Methods

Assignment 2 (Individual) 2024

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

20 %

Managing Code

You are expected to manage your source code. A GitLab group has been created for each student, and these groups must be used to create the project to be submitted:

<https://gitlab.ecs.vuw.ac.nz/course-work/swen301/2024/assignment-2/{username}>

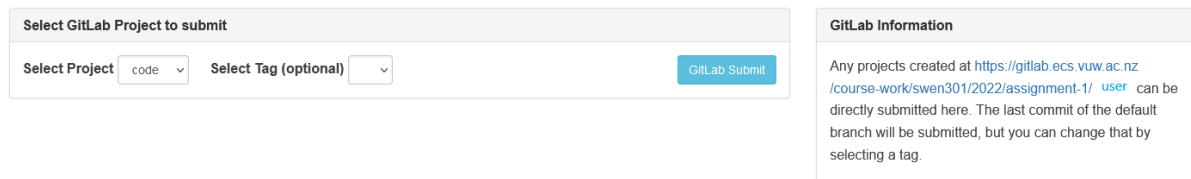
We expect that the repository is actually used, i.e. you should frequently commit. One final commit at the end is not acceptable and will be penalised.

The use of public external repositories (GitHub, Bitbucket etc) is not permitted, and if detected (using repository search and duplications with submissions detected via Moss) will be penalised.

You are expected to 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 (**convention over configuration**) is part of the discussion in this course. For instance, in task 2, 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 it is possible to have the same package for main and test code in different folders -- **src/main/java** and **src/test/java**).

What and How to submit

There is no file upload, submission is via the gitlab-assessment system integration. It is highly recommended to create a tag for submission. In the submission system, you will see the following dialog:



Select GitLab Project to submit

Select Project `code` Select Tag (optional) **GitLab Submit**

GitLab Information

Any projects created at <https://gitlab.ecs.vuw.ac.nz/course-work/swen301/2022/assignment-1/> user can be directly submitted here. The last commit of the default branch will be submitted, but you can change that by selecting a tag.

On submission, the project will be cloned, checked out (last commit in main or selected tag if available), and submitted.

You can (and should) verify what exactly you have submitted – the submission system will create a tag named **engr-submit-`<timestamp>`** for your submission.

It is expected that you verify your submission as follows:

- 1. clone the repo into a new folder**
- 2. checkout this tag created by the submission system (`engr-submit-<timestamp>`)**
- 3. build your project with maven (not the IDE) and ensure that the build succeeds (compilation succeeds, tests pass)**

If you don't do this and submit the wrong code or the wrong version, the submission might be marked zero !

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.

Prepare

Do the log4j tutorial and attend or watch the lectures on log4j.

There are also “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 to be extended is log4j, and its purpose and architecture has been discussed in the lectures, for additional resources check the **Prepare** section. The tasks focus on writing a memory appender. An appender is “where the logs go” -- examples are the console (logs go to **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 does not use I/O to append logs, 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 17, 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 17 requirement (for both compilation source and target) in the Maven using the **<properties>** elements (i.e., in the POM). **[1 mark]**
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](https://gson.org), [jackson](https://jackson.org), [fastjson](https://fastjson.org), [jsonp](https://jsonp.org). Use a recent version of one of those

libraries not flagged as vulnerable on the mvnrepository.com website. 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.

Note that for each log event, a JSON string representing a valid JSON object (i.e., “{ . . }”) is to be produced. This does not mean that the JSON written to a file with a file appender using this layout is valid JSON as this misses the commas separating JSON objects, and the square brackets for the outer array. This is similar to the HTMLAppender discussed in the lecture – it produces HTML snippets, but not full HTML elements. While the [header and footer methods](#) could be used to create JSON arrays, this won't solve the issue of not having commas separating JSON objects within arrays. **[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 instruction coverage of 80% or better for the class **JsonLayout**. Use annotation-based JUnit5 tests. **[2 marks]**
4. In this project, create a log4j appender **nz.ac.wgtn.swen301.assignment2.MemoryAppender** as a Java class:
 - a. **MemoryAppender** must have a public zero-argument constructor, and allow access to properties using getters and setters. Setters are only required for the **name** and **maxSize** properties. **[1.5 mark]**
 - b. **MemoryAppender** 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]**
 - c. **MemoryAppender** have a **name** property¹ of type **String**. **[0.5 mark]**
 - d. Logs can be accessed using the following non-static method:
java.util.List<LoggingEvent> getCurrentLogs() **[1 mark]**
 - e. the list returned by **getCurrentLogs()** must not be modifiable **[1 mark]**
 - f. **MemoryAppender** 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. **[1 marks]**
 - g. **MemoryAppender** has a method to export stored log events to a JSON file: **void export(String fileName)**. This method exports the log events currently stored into a JSON file. The JSON file should contain a JSON array 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 may use the **JsonLayout** described earlier. The file name passed should support relative file names relative to

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.

the application working directory. I.e. if the file name is out.json, the file is created in the working directory of the application (when running this from an IDE, this is the project root folder). Do not prepend paths to the file names passed as argument ! **[1 marks]**

5. Write JUnit tests in a class **nz.ac.wgtn.swen301.assignment2.MemoryAppenderTest** that test **MemoryAppender** in combination with different loggers and levels. Aim for high instruction coverage of 80% or better for the class **MemoryAppender**. Use annotation-based JUnit5 tests. **[2 marks]**
6. Create an MBean object for each instance of the **MemoryAppender** 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 currently stored logs (excluding discarded 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 export(String fileName)**
7. Implement a class **nz.ac.wgtn.swen301.assignment2.example.LogRunner** that 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. Integrated the jacoco plugin into the Maven build pom.xml in order to automate the measurement of test coverage. Running **mvn clean test jacoco:report** must create a CSV-formatted coverage report in **target/site/jacoco/jacoco.csv** . **[1 mark]**
9. Create a report in **README.md** in markdown format that contains the following sections:
 - a. Any special Instructions if needed
 - b. A discussion of 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, ...). Compare your choice against two alternatives (table with pros , cons), add links for evidence where appropriate. **[1 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). See task 7 – you can run this executable (from an IDE) to test the MBean by inspecting it.

JSON Export Format

This is an example of the JSON format to be produced by the export function -- only the following 5 attributes (logger) name, level, timestamp, thread and message need to be represented, note that **all property names are lower-case** -- this is part of the requirements ! Those correspond to properties in [LoggingEvent](#).

Formatting details such as indents, new lines etc are not important. The order of properties in a JSON object is also not important. In particular, pretty-printing is recommended, but not required. The timestamp format to be used is defined as in [java.time.format.DateTimeFormatter::ISO_INSTANT](#)

```
[
  {
    "name": "foo",
    "level": "WARN",
    "timestamp": "2011-12-03T10:15:30Z",
    "thread": "main",
    "message": "something went wrong"
  },
  {
    "name": "foo",
    "level": "DEBUG",
    "timestamp": "2011-12-03T10:15:30Z",
```

```
        "thread": "main",  
        "message": "interesting !"  
    }  
]
```

Resources on JMX / MBeans

<https://docs.oracle.com/javase/tutorial/jmx/mbeans/standard.html>

<https://www.baeldung.com/java-management-extensions>

Change History

Date	Description
28 Aug 24 , 9 Sept 24	Removed the condition that the jar should be executable from the submission verification procedure