Massey University

159.251 - Software Design and Construction

Assignment 2 (22%)

Deadlines

You must submit your final work using the stream submission system no later than **Sunday 23 Oct 2022.** The penalty is 10% deducted from the total possible mark for every day delay in submission (one day late – out of 90%, two days late then out of 80% … etc.).

You are expected to manage your source code, this includes making frequent backups. It is strongly recommended (but not required) to use a **private git repository** for this assignment, and commit as frequently as possible. "*The Cat Ate My Source Code*" is not a valid excuse for a late submission.

How to submit

1. Upload a zip file consisting of:
    a. The Maven project folder (inc. **pom.xml**)
    b. **performance-analysis.pdf** -measure time and memory consumption
    c. **coverage.pdf/html** - the pdf *or* html version of the coverage report created by Maven
2. upload this file to stream - note: the max upload size is set to 20 MB
3. verify the submission: download the zip file, unzip it into a new folder and inspect content, run Maven from the command line, check the output including generated jar files

Task

Work **individually** to create the following program in Java.

Create a project assign251_2 using the Maven project layout, and within this project, create a project that implements custom appender and layout objects for Log4j. For this, you will need to create an appender and layout **that work with the other Log4j objects** (i.e. implementing relevant Log4j abstract classes or interfaces), test them and run profiling tools on them to gauge their correctness and efficiency.

Note, there is no main class for this project, it will be run via your tests from sections 3 and 4.

You may want to consider a test-driven development methodology, where your first step is to start with section 3 and work backwards. This will allow you to check that your classes are working correctly as you go.

1. Implement a log4j appender assign251_2.MemAppender [7 marks]

In this task, you will need to implement a custom log4j appender, which can be used directly with the log4j logger. This **MemAppender**, unlike normal appenders, stores logs in memory and prints them on demand. There is a limit to how many log events will be kept in memory (this should be configurable), and if the maximum is reached, the oldest logs should be deleted.

Implementation details:

- It enforces the **singleton** pattern.
- It stores the LoggingEvents in a list. This is supplied by dependency injection (note: if you have already created a default, that is okay).
- It will need a layout. This will need to be able to be supplied when the MemAppender is collected, and via the **setLayout()** method. If a layout is not supplied, and code calling it is needed, appropriate precondition checks should be used (as some code may not use the appender with the layout, so it is a valid option not to supply one, as long as you don't use any functionality that requires it).
- There are three ways to get information about the LoggingEvents that it stores:
    a. Call the method **getCurrentLogs()** which will return an **unmodifiable** list of the LoggingEvents.
    b. Call the method **getEventStrings()** which will return an **unmodifiable** list of strings (generated using a layout stored in the MemAppender).
    c. Call the method **printLogs()** which will print the logging events to the console using the layout and then clear the logs from its memory.
- It has a property called **maxSize**, which needs to be configurable. When this size is reached, the oldest logs should be removed to make space for the new ones.
- The number of discarded logs should be tracked, and can be accessed using **getDiscardedLogCount()**. This should be stored as a **long** type, as there may be many discarded logs.

Note: Be careful to observe the DRY principle - there are overlapping requirements above.

3.5 marks	Correct implementation of the singleton pattern and dependency injection options for the list and layout.
2 marks	Correct implementation of the information printing / collection methods,

	along with sensible precondition checks where appropriate.
1.5 mark	Correct implementation of maxSize and associated features.

2. **Implement a layout assign251_2.VelocityLayout [3 marks]**
    a. **VelocityLayout** basically works like **PatternLayout**, but uses Velocity as the template engine. This layout should work with log4j appenders as well as the MemAppender.
    b. Variable to be supported:
        i. c (category)
        ii. d (date using the default toString() representation)
        iii. m (message)
        iv. p (priority)
        v. t (thread)
        vi. n (line separator)
    c. This means that the variable syntax is different, e.g. use **$m** instead of **%m**
    d. VelocityLayout should have options to set its pattern both in the constructor and via a setter. An example string pattern could look like:
       `“[$p] $c $d: $m”`
3. **Write tests that test your appender and layout in combination with different loggers, levels and appenders [4 marks]**
    a. Use JUnit for testing your appender and layout. Aim for good test coverage and precise asserts.
    b. Use the tests to show both the appender and layout working with different combinations of built-in log4j classes as well as with each other.
    c. Tests should be stored in the appropriate locations according to the Maven folder structure.
4. **Write tests to stress-test your appender/layout by creating a large amount of log statements [6 marks]**
    a. Create a separate test class for stress tests.
    b. Use these tests to compare the performance between MemAppender using a LinkedList, MemAppender using an ArrayList, ConsoleAppender and FileAppender - measure time and memory consumption (using JConsole or VisualVM or any profiler)
    c. Consider how to output your logs in such a way that makes comparisons between the MemAppender and other appenders sensible.
    d. Use these scripts to compare the performance between **PatternLayout** and **VelocityLayout**
    e. Stress tests should test performance before and after **maxSize** has been reached, and with different maxSize values.
        i. parameterised tests may be helpful here.
    f. Write a short report summarising your findings (embed screenshots of memory usage charts in this report taken from VisualVM). The report name should be **performance-analysis.pdf**
    g. Measure your test coverage of the written tests by generating branch and statement coverage reports using Jacoco or Emma. Submit this report with your

project (should be placed under ~/target/ folder"

Note that the marks for this section will be based on your reporting, the effectiveness of your stress tests in probing into the efficiency of the classes, and the overall integration testing, checking that these classes work in combination with other relevant out-of-the-box classes.

5. Write a Maven build script [2 marks]

a. The Maven script should be used to build the project including compiling, testing, measuring test coverage, and dependency analysis. All dependencies should be managed with your maven build.
b. Use the jacoco Maven plugin for measuring test coverage.

Hints

- You can use any development environment you prefer, as it is a Maven project.
- Library approved list: only the following libraries can be used: Apache log4j, Apache Velocity, JUnit 5, Google Guava, Apache Commons Collections, JaCoCo (for code coverage).

Penalties

1. Code that is not self-documenting, or long or complex methods.
2. Violating the Maven standard project layout or Java naming conventions.
3. Use of absolute paths (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 not on the whitelist

Bonus Question [2 marks]

You can get 100% for the assignment without this. This will give you additional marks up to the maximum if you lose some elsewhere.

Create an MBean object for each instance of the **MemAppender** to add JMX monitoring to this object, the properties to be monitored are

1. the log messages as array
2. the estimated size of the cached logs (total characters)
3. the number of logs that have been discarded

Assessment

Your assessment will be based on the following criteria:

Criteria	Mark
Implementation of log4j appender assign251_2.MemAppender	**7**
Correct implementation of the singleton pattern and dependency injection options for the list and layout.	3.5
Correct implementation of the information printing / collection methods, along with sensible precondition checks where appropriate	2
Correct implementation of maxSize and associated features.	1.5
Implementation of layout assign251_2.VelocityLayout	**3**
Correct use of the Velocity template engine	1
Works with appenders and MemAppender	1
Supports listed variables	1
Testing the implemented appender and layout	**4**
Use of Junit with good coverage and precise asserts	2
Tests show that the appender and layout work with different combinations of built-in log4j classes and each other	1.5
Tests stored in appropriate locations following Maven directory structure	0.5
Stress-testing your appender/layout	**6**
Separate class for stress tests	0.5
Comparison of performance between MemAppender using LinkedList, ArrayList, ConsoleAppender and FileAppender - with measurements: time, memory consumption for different maxSizes	2
Scripts to compare velocity and pattern layout	1
Report of stress test findings with an analysis of the stress test results and measurements	2
Test coverage reports	0.5
Build management	**2**

Uses maven for dependency, coverage (using jacoco)	2
(extra/bonus) Implementation of an MBean object for instances of MemAppender for JMX monitoring of properties: log messages, estimated size of cached logs, number of logs discarded	up to 2 marks
Total	**22 (max)**