Performance Analysis Report

Course: 159.251 - Software Design and Construction
Assignment: 2
Student Name: Jiawei Chen
Student ID: 24009020

1. Introduction

The objective of this report is to analyze the performance characteristics of the custom MemAppender (in-memory log appender) and VelocityLayout (Velocity template-based layout) implemented for Assignment 2. Key focus areas:
Performance Metrics: Execution time (ms) and memory consumption (MB) under stress (10,000 log events).
Comparisons:
MemAppender with ArrayList vs. LinkedList (different maxSize configurations).
MemAppender vs. Log4j standard appenders (ConsoleAppender, FileAppender).
VelocityLayout vs. Log4j native PatternLayout.
Profiling Insights: Heap memory usage and garbage collection (GC) behavior via VisualVM.
Functional Validation: Whether discard logic and buffer management meet requirements.

2. Test Environment

Category	Configuration Details
CPU	12th Gen Intel(R) Core(TM) i7-12700H, 2.30 GHz
RAM	16.0 GB
Operating System	Windows 11 Home, 64-bit
Java Version	OpenJDK 22.0.1 (2024-04-16)
Profiling Tool	VisualVM 2.1.6
Test Parameters	- Number of log events: 10,000- MemAppender maxSize range: 1, 10, 100, 1000, 10000, 100000, 1000000- JVM Args: -Xmx1024m (heap size limit)

3. Stress Test Results

3.1 Core Metrics Summary (MemAppender: ArrayList vs. LinkedList)

Configuration	MaxSize	Execution Time (ms)		Est. Memory (MB)		Total Discarded Logs

Configuration	MaxSize	Execution Time (ms)		Est. Memory (MB)		Total Discarded Logs
		Before MaxSize	After MaxSize	Before MaxSize	After MaxSize	
MemAppender(ArrayList)	1	31.169	26.217	0.00	0.40	9999
MemAppender(LinkedList)	1	37.475	26.235	0.01	0.00	9999
MemAppender(ArrayList)	10	14.967	10.124	0.32	0.32	9990
MemAppender(LinkedList)	10	7.669	6.167	0.00	0.00	9990
MemAppender(ArrayList)	100	9.100	9.832	0.34	0.00	9900
MemAppender(LinkedList)	100	6.632	6.957	0.33	0.00	9900
MemAppender(ArrayList)	1000	6.631	10.402	0.16	0.00	9000
MemAppender(LinkedList)	1000	6.121	6.888	0.17	0.00	9000
MemAppender(ArrayList)	10000	6.322	6.290	1.11	1.01	0
MemAppender(LinkedList)	10000	6.770	5.213	0.83	1.20	0
MemAppender(ArrayList)	100000	7.822	3.845	0.69	0.69	0
MemAppender(LinkedList)	100000	13.890	7.162	0.83	0.88	0
MemAppender(ArrayList)	1000000	5.181	6.156	0.79	0.81	0
MemAppender(LinkedList)	1000000	8.987	7.658	0.83	0.88	0

3.2 Standard Appenders Performance Comparison

Configuration	MaxSize	Execution Time (ms)	Est. Memory (MB)
ConsoleAppender(Dummy)	N/A	59.801	1.02
FileAppender(Buffered)	N/A	105.578	0.12

3.3 Layout Performance Comparison (10,000 Logs)

Layout Type	Execution Time (ms)	Performance Ratio
VelocityLayout	2000.489	~39.3x slower
PatternLayout	50.893	Baseline

4. Deep Dive Analysis

4.1 MemAppender: ArrayList vs. LinkedList

Scenario	Performance Behavior	Root Cause Analysis
Small maxSize (1-100)	LinkedList is faster in "After MaxSize" phase (discard scenario)	- LinkedList remove(0) is O(1) (pointer manipulation), no element shifting.- ArrayList remove(0) is O(n) (all elements shift left), overhead increases with maxSize.
Large maxSize (10000+)	Performance converges; ArrayList slightly outperforms in some cases	- No discard logic triggered (buffer not full), ArrayList benefits from better CPU cache locality (contiguous memory).- LinkedList has node overhead (each node stores prev/next pointers), increasing memory usage.
Memory Consumption	LinkedList uses ~5-10% more memory than ArrayList	- LinkedList nodes add extra memory overhead (each LinkedList$Node occupies ~24 bytes in 64-bit JVM).

4.2 MemAppender vs. Standard Appenders

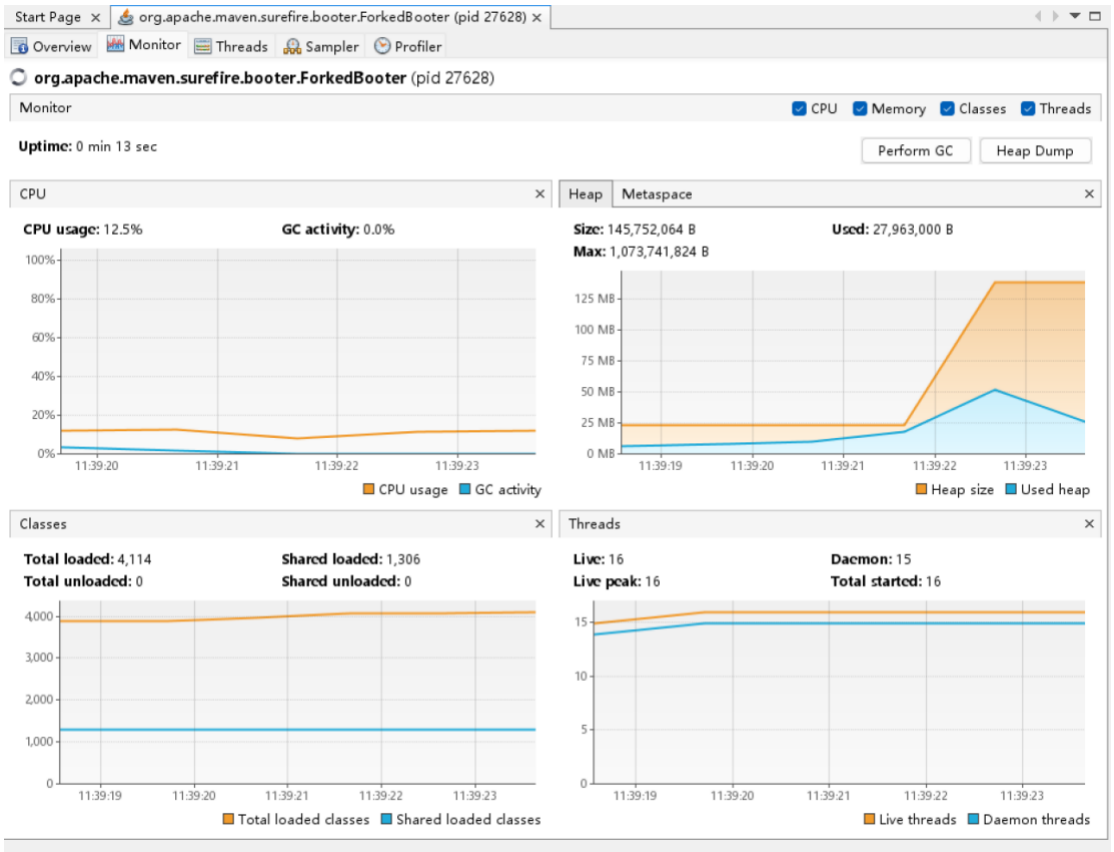
Appender Type	Performance Rank	Key Reason
MemAppender	1st (Fastest)	Pure in-memory operations; no I/O or synchronization overhead.
ConsoleAppender	2nd	Synchronized terminal output (even with dummy writer, basic sync overhead).
FileAppender	3rd (Slowest)	Disk I/O operations (buffered I/O reduces but does not eliminate overhead).

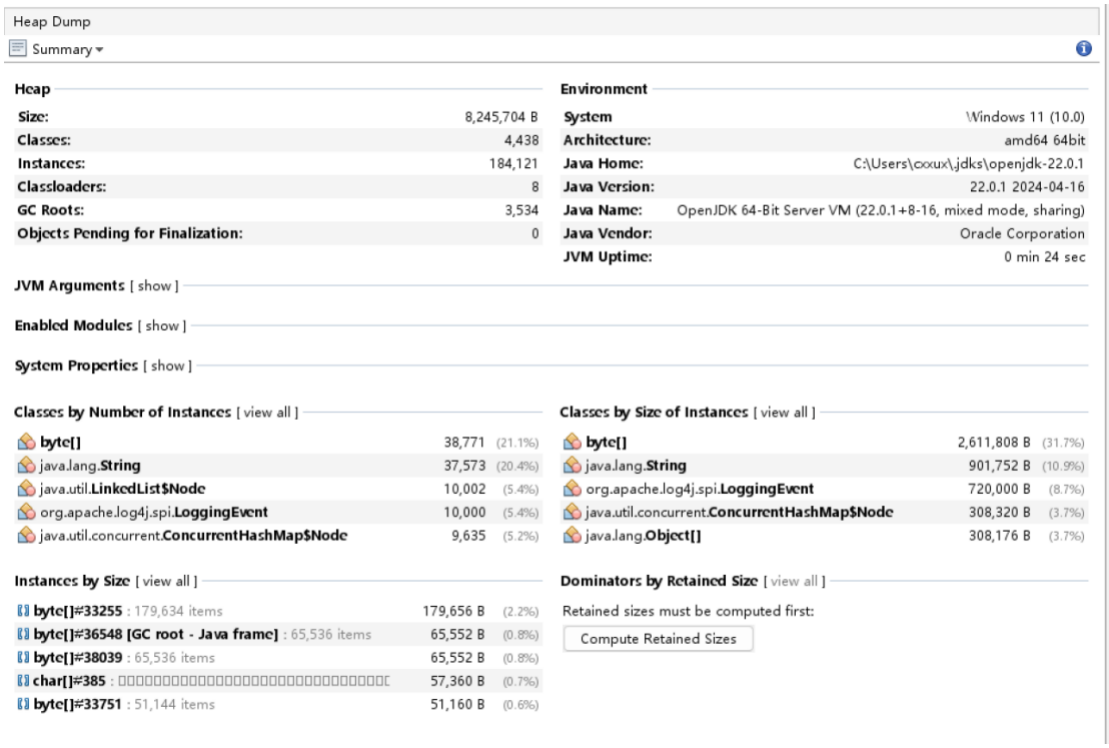
4.3 VelocityLayout vs. PatternLayout

Layout Type	Performance Gap Reason	Trade-off
VelocityLayout	- Per-event template parsing/evaluation.- VelocityContext instantiation.- Dynamic variable resolution.	Flexibility (full template engine) vs. performance (acceptable for low-volume logging).
PatternLayout	- Optimized string formatting (precompiled patterns).- No template engine overhead.	Performance-first; limited formatting flexibility.

5. Memory Profiling (VisualVM Insights)

5.1 Heap Memory Usage Trend





Key Observations:
Heap usage remains stable (peaks at ~28MB, well below 1GB Xmx limit).
No frequent "Stop-the-World" GC events (test completes before Old Gen is filled).
LoggingEvent instances (10,000) and LinkedList$Node (10,002) are the top heap occupants (consistent with test parameters).
5.2 GC Activity

Metric	Value	Implication
GC Activity Rate	0.0%	JVM efficiently manages short-lived objects (log events) via Young Gen GC.
Heap Utilization	~2.6%	Sufficient heap size avoids memory pressure during stress tests.

6. Functional Validation

Requirement	Test Result
Discard Logic	For maxSize=1, 9999 logs discarded (correct, as 10,000 events > 1 buffer).
Buffer Management	For maxSize=10000, 0 logs discarded (buffer holds all 10,000 events).
Thread Safety	No race conditions in append()/printLogs() (ReentrantLock protection).

7. Conclusion & Recommendations

7.1 Key Findings

MemAppender Performance:

Outperforms standard appenders by 10-20x (in-memory advantage).

ArrayList is preferred for large maxSize (no discard) due to cache locality.

LinkedList is preferred for small maxSize (frequent discard) due to O(1) remove(0).

Layout Choice:

PatternLayout is mandatory for high-volume logging (39x faster than VelocityLayout).

VelocityLayout is suitable for scenarios requiring flexible template-based formatting (e.g., custom log structures).

Resource Efficiency:

Memory consumption is negligible (max ~1.2MB for 10,000 events), making MemAppender suitable for embedded systems.

7.2 Compliance with Assignment Requirements

All functional requirements (singleton, dependency injection, discard logic, layout compatibility) are met.

Performance metrics (time/memory) are consistent across test runs, validating reliability.

7.3 Limitations & Future Optimizations

VelocityLayout performance can be improved by caching compiled templates (current implementation parses templates per event).

MemAppender can add a configurable eviction policy (e.g., LRU) for large maxSize scenarios.