Lab 3 Report – Framework Testing

1. Introduction

The goal of Lab 3 was to expand our benchmarking framework by implementing and testing advanced timing features,
improving output readability, and adding user-level control through benchmark cancelation. The tasks built upon the
foundational structure from Lab 1 and Lab 2, focusing on usability, precision, and flexibility of the framework.

2. Implemented Tasks

2.1 Implementing a Demo Testbench

A demo benchmark class (DemoBenchmark.java) was created to implement the IBenchmark interface. It includes:
- initialize(Object... params): creates a randomly-filled integer array.
- run(): performs a bubble sort algorithm on the array.
- clean(): clears resources.
- cancel(): sets an internal running flag to false to gracefully stop execution.

A test class (TestDemoBench.java) runs the benchmark multiple times while measuring each iteration’s execution time using pause-resume logic.

2.2 Timer Functionality

A Timer class was implemented with:
- start(), pause(), resume(), stop(): to measure both individual and cumulative time segments.

The functionality was tested in two ways:
- Using Thread.sleep(n) in a controlled test (TestTimerSleep.java) to compare expected and measured delays.
- Running benchmark in a loop with pause/resume to simulate partial benchmarking and accumulate timing (TestDemoBench.java).

2.3 Time Unit Conversion

To improve readability, a TimeUnit enum was implemented in the logging package, with options:
- Nano, Micro, Milli, and Sec

A writeTime(String, long, TimeUnit) method was added to the ILogger interface and implemented in ConsoleLogger, allowing formatted output in the desired time unit.

Example output:
```
Iteration 1 time: 2.103 ms
Total benchmark time: 10.591 ms
```

2.4 Canceling a Benchmark

The framework was extended to support canceling a running benchmark. Inside the DemoBenchmark class:
- A running boolean field was used.
- cancel() sets running to false.
- run() checks running in each loop iteration.

This was tested using a separate thread in TestCancelBench.java, where the benchmark was canceled after 100ms of execution using Thread.sleep and cancel().

3. Code Structure

```
src/
├── benchmark/
│   ├── IBenchmark.java
│   └── DemoBenchmark.java
├── logging/
│   ├── ILogger.java
│   ├── ConsoleLogger.java
│   └── TimeUnit.java
├── timing/
```

```
│   └── Timer.java
└── testbench/
    ├── TestDemoBench.java
    ├── TestCancelBench.java
    └── TestTimerSleep.java
```

4. Results and Testing
All functionalities were successfully tested:
- Accurate timing and partial measurement using pause/resume
- Cancelation without crashing or interrupting the application
- Flexible output formatting in milliseconds and seconds
Console outputs confirmed consistent timing and correct implementation of each required method.

5. Conclusion
This lab finalized a fully functional benchmarking framework in Java, with extensible benchmark logic, precise timing control,
readable output formatting, and user-level cancelation. The structure is clean, modular, and ready to support future benchmarks and logging features.