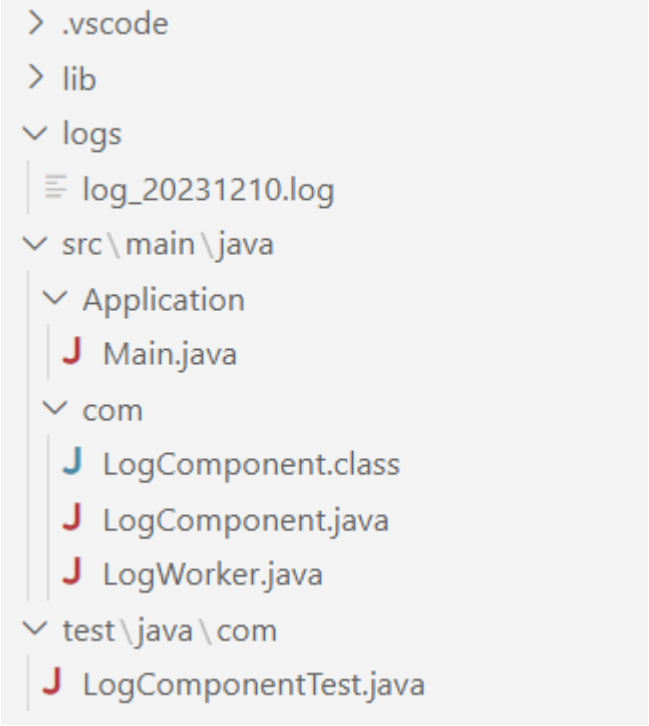
Aneesh N A - aneeshnasmg6@gmail.com

DESCRIPTION

In this example, I've created a basic LogComponent using a separate thread for asynchronous logging. This component writes logs to daily log files in the specified directory. The write method is non-blocking, allowing the calling application to continue its work without waiting for logs to be written.
This

PROJECT STRUCTURE



The calling is being done from the application package, achieving independence and modularity.
Main components are LogComponent.java and LogWorker.java

TESTS

LogComponentTest.java is being implemented as the unit test class.
Unit tests are being implemented as separate functions.

SOLID Principles -

1. **Single Responsibility Principle (SRP):**

A class should have only one reason to change, meaning that a class should have only one job or responsibility. LogComponent and LogWorker classes perform single functionalities.

2. **Open/Closed Principle (OCP):**

Any new functionality should be added through the creation of new classes, rather than by modifying existing classes. This allows for easy extension without altering existing code.

3. **Liskov Substitution Principle (LSP):**

Objects of a superclass should be replaceable with objects of a subclass without affecting the correctness of the program.

.

4. **Interface Segregation Principle (ISP):**

Clients should not be forced to depend on interfaces they do not use.

5. **Dependency Inversion Principle (DIP):**

High-level modules should not depend on low-level modules. Both should depend on abstractions.Abstractions should not depend on details; details should depend on abstractions. I've achieved modularity by using independent classes.

Due to lack of time, the same architecture as that of ***src*** could not be implemented in the ***test***.