MSCS-632: Assignment 6: Lab Report: Data Processing System Implementation in Java and Go

---

Sandesh Pokharel

University of the Cumberlands

MSCS-632 Advanced Programming Languages

Dr. Vanessa Cooper

April 15, 2025

Introduction

This report presents the design and implementation of a concurrent Data Processing System developed in two different programming languages: Java and Go. The purpose of this system is to simulate multiple worker threads that process tasks from a shared queue, log the results, and demonstrate proper synchronization, error handling, and logging techniques. The implementation covers all requirements including task queuing, concurrency management, safe logging, and graceful termination.

Github link

The link to the Github repository is given here:
https://github.com/sanspokharel26677/MSCS-632-Assignment6

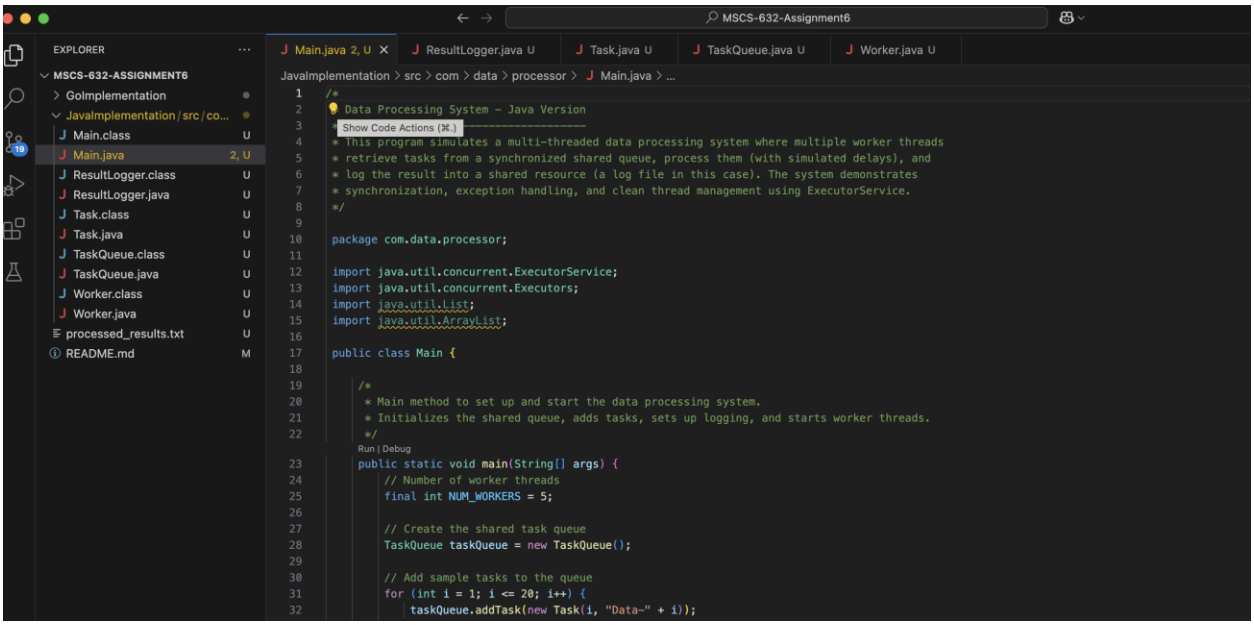
Java Implementation

The Java version uses `ExecutorService` to manage worker threads and a synchronized custom queue for tasks. Each worker thread retrieves tasks, processes them with a simulated delay using `Thread.sleep()`, and logs the result to a shared output file using a synchronized `ResultLogger` class.

Key features include:

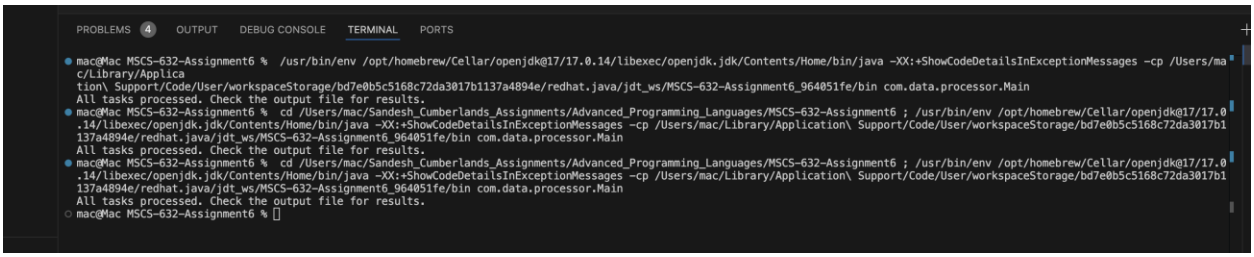
- - Thread pool management using `Executors`
- - Safe task retrieval with `synchronized`, `wait()`, and `notify()`
- - Graceful thread shutdown
- - Exception handling using `try-catch` blocks

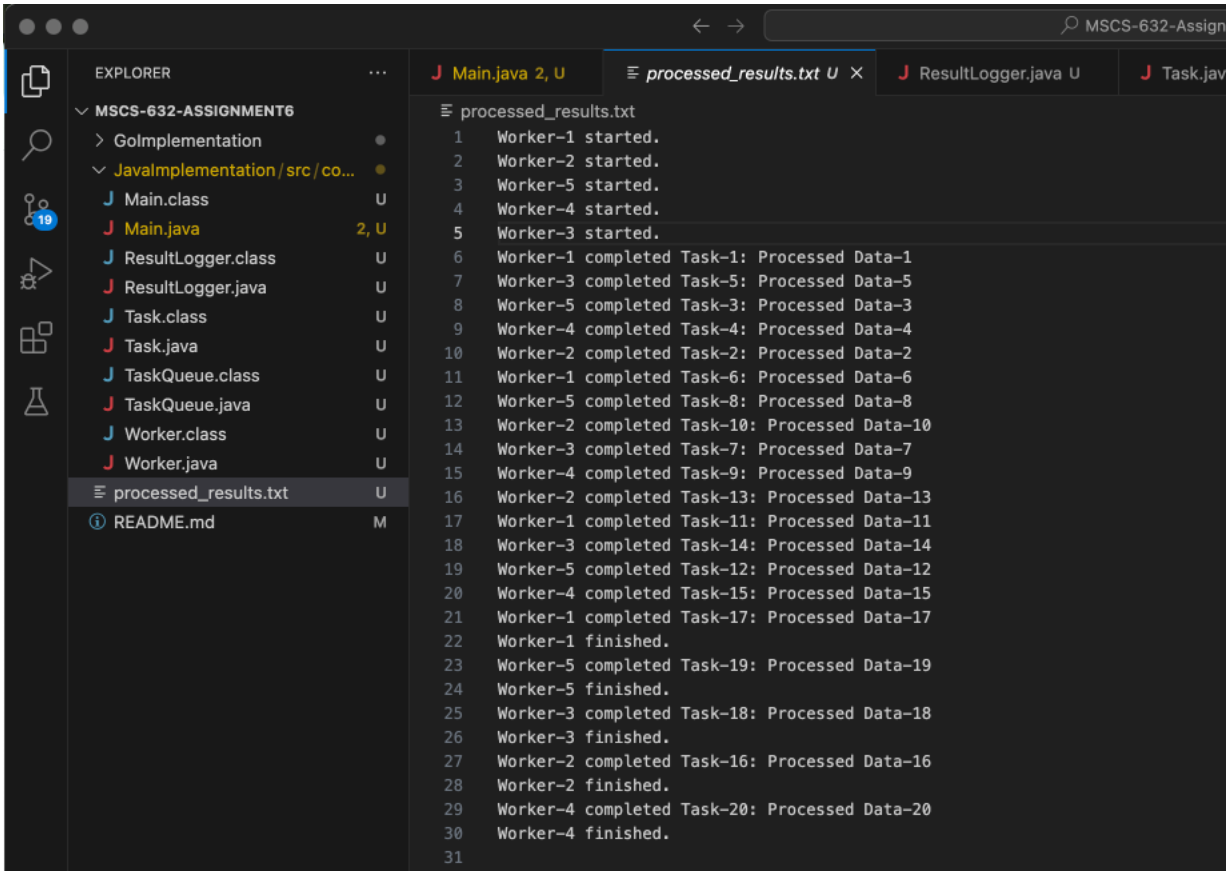
📸 Screenshot: *Java project structure in IntelliJ*




```
/*
 Data Processing System - Java Version
 ------------------------------------
* This program simulates a multi-threaded data processing system where multiple worker threads
* retrieve tasks from a synchronized shared queue, process them (with simulated delays), and
* log the result into a shared resource (a log file in this case). The system demonstrates
* synchronization, exception handling, and clean thread management using ExecutorService.
*/

package com.data.processor;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.List;
import java.util.ArrayList;

public class Main {

    /*
     * Main method to set up and start the data processing system.
     * Initializes the shared queue, adds tasks, sets up logging, and starts worker threads.
     */
    public static void main(String[] args) {
        // Number of worker threads
        final int NUM_WORKERS = 5;

        // Create the shared task queue
        TaskQueue taskQueue = new TaskQueue();

        // Add sample tasks to the queue
        for (int i = 1; i <= 20; i++) {
            taskQueue.addTask(new Task(i, "Data-" + i));
```

📸 Screenshot: *Java output in processed_results.txt*





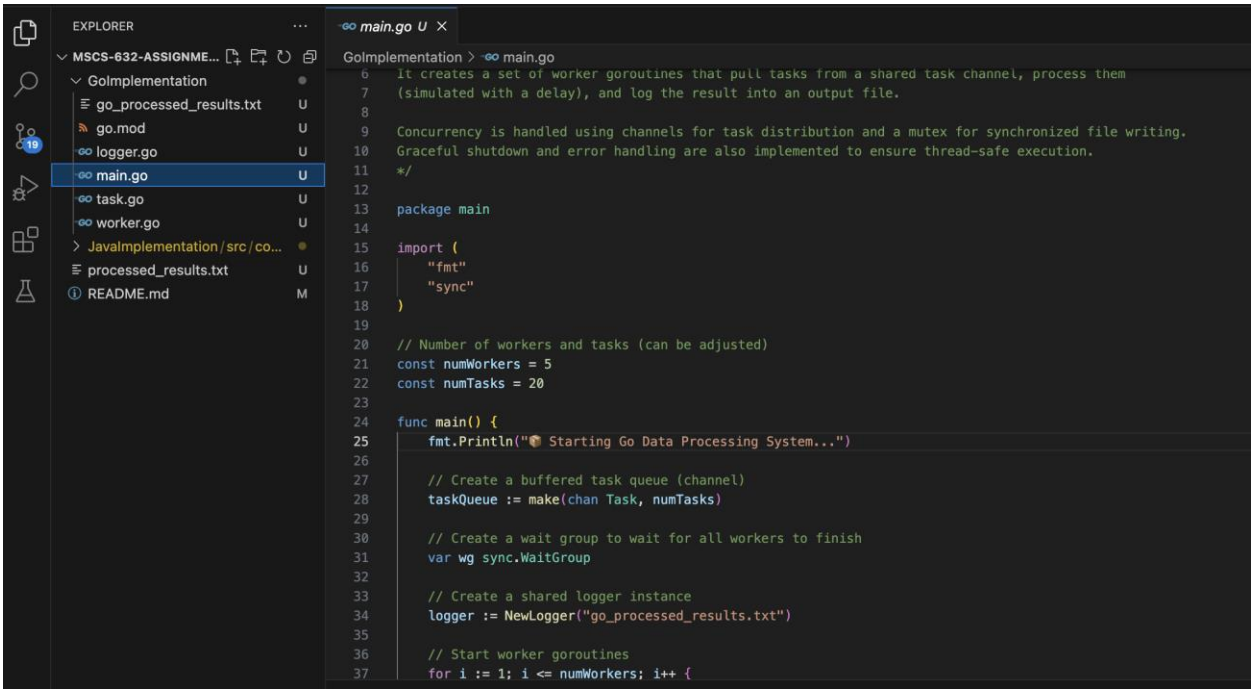
Go Implementation

The Go version uses goroutines and buffered channels for concurrency. Each goroutine acts as a worker and retrieves tasks from a channel. Logging is done via a `Logger` struct with a mutex to ensure thread-safe file access. The system is initialized with a `main.go` file that sets up task dispatch and worker routines.

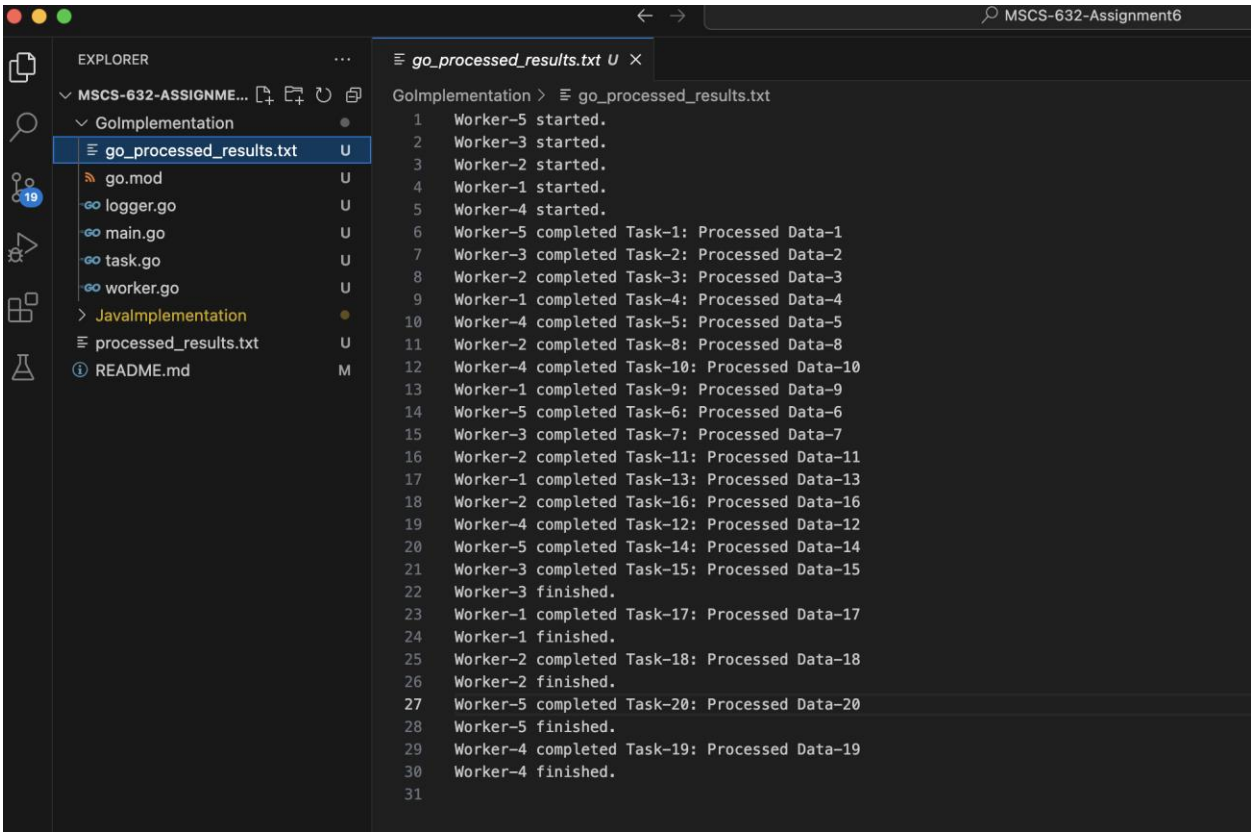
Key features include:

- - Goroutines for lightweight concurrency
- - Buffered channels as a thread-safe task queue
- - Panic recovery using `defer` and `recover()`
- - Synchronized logging using `sync.Mutex`

📸 Screenshot: *GoImplementation folder with .go files*



📸 Screenshot: *Go output in go_processed_results.txt*



Java vs Go: Concurrency and Error Handling

The key differences between Java and Go implementations are as follows:

- - Java uses heavyweight threads managed via ExecutorService; Go uses lightweight goroutines.
- - Java uses synchronized blocks and locks; Go relies on channels and mutexes.
- - Java handles exceptions with try-catch; Go uses error returns and panic recovery.

- Logging is handled via synchronized writers in Java; in Go, mutexes guard file writes.

Output Validation

Both implementations were tested and verified. All 20 tasks were processed exactly once in both cases, in a non-sequential order, confirming parallelism. Output files were clean, without data corruption, and worker start/finish logs confirmed graceful termination.

Conclusion

This project successfully demonstrates two concurrency models and their real-world application in building a multi-threaded data processing system. By comparing Java and Go, this assignment highlights how language-level constructs impact synchronization, thread management, and error handling.