**Assignment 6: Multi-threaded Data Processing System**

Student Name

University

Advanced Programming Languages - Full term  
professor Name

July 31, 2025

**Ride Sharing System - Concurrency and Exception Handling in Java and Go**

**Introduction**

The Ride Sharing System is a concurrent data processing model designed to mimic multiple worker threads or goroutines created to run in parallel. Architecture consists of a common queue of measures and a common results resource in which results are transferred. The primary objective of the system is to demonstrate proper concurrency aspects, synchronization, exception handling, and logging. The implementation is in both Java and Go, two languages that have different concurrency models. The parallel implementation of both languages in this manner affords a comparative insight into how both languages handle parallel computation and the safety of resources.

**Concurrency and Exception Handling in Java**

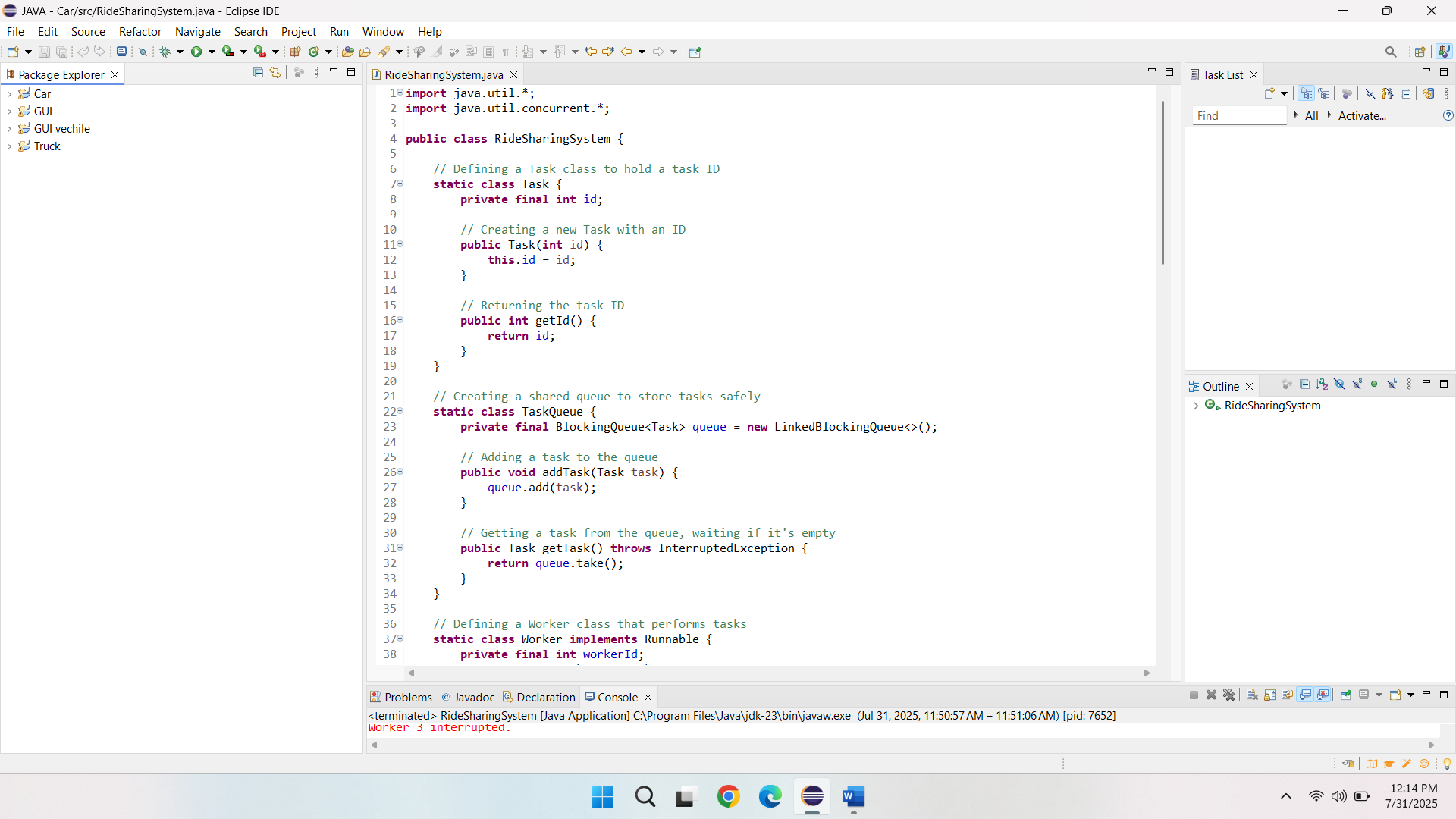
Java has a thread-based model of concurrency. The ExecutorService is used to create a fixed-size thread pool, which manages the lifetime of the worker threads in the most efficient way. Tasks are saved in a blocking queue to achieve thread native access and avoid conflicts of data between the different threads. Both workers pull a task and work on it by simulating a delay, then writing the output to a common list (Khanna et al., 2023). The result list is secured by a synchronized block in case the mutual exclusion is required during update. There is structured error handling. Interrupted Exception will be dealt with to deal with disruptions to threads, and other unanticipated mistakes will be handled with general error blocks. Summarily, Logging is used to account for task progress and employee activity, where error reporting is also implemented. Such a strategy guarantees the completion of execution of all threads, along with the loss of data or deadlocks.

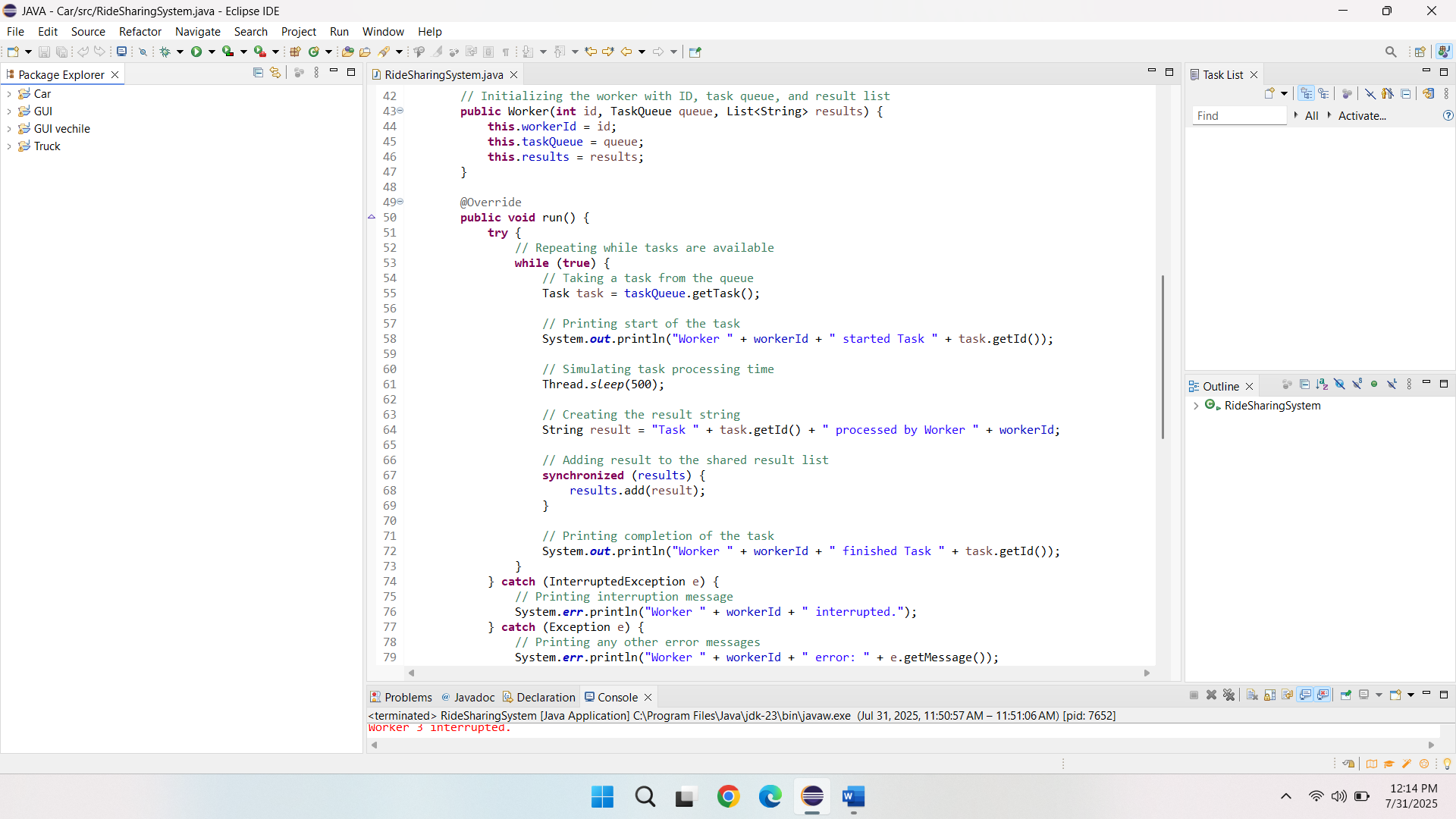
**Concurrency and Exception Handling in Go**

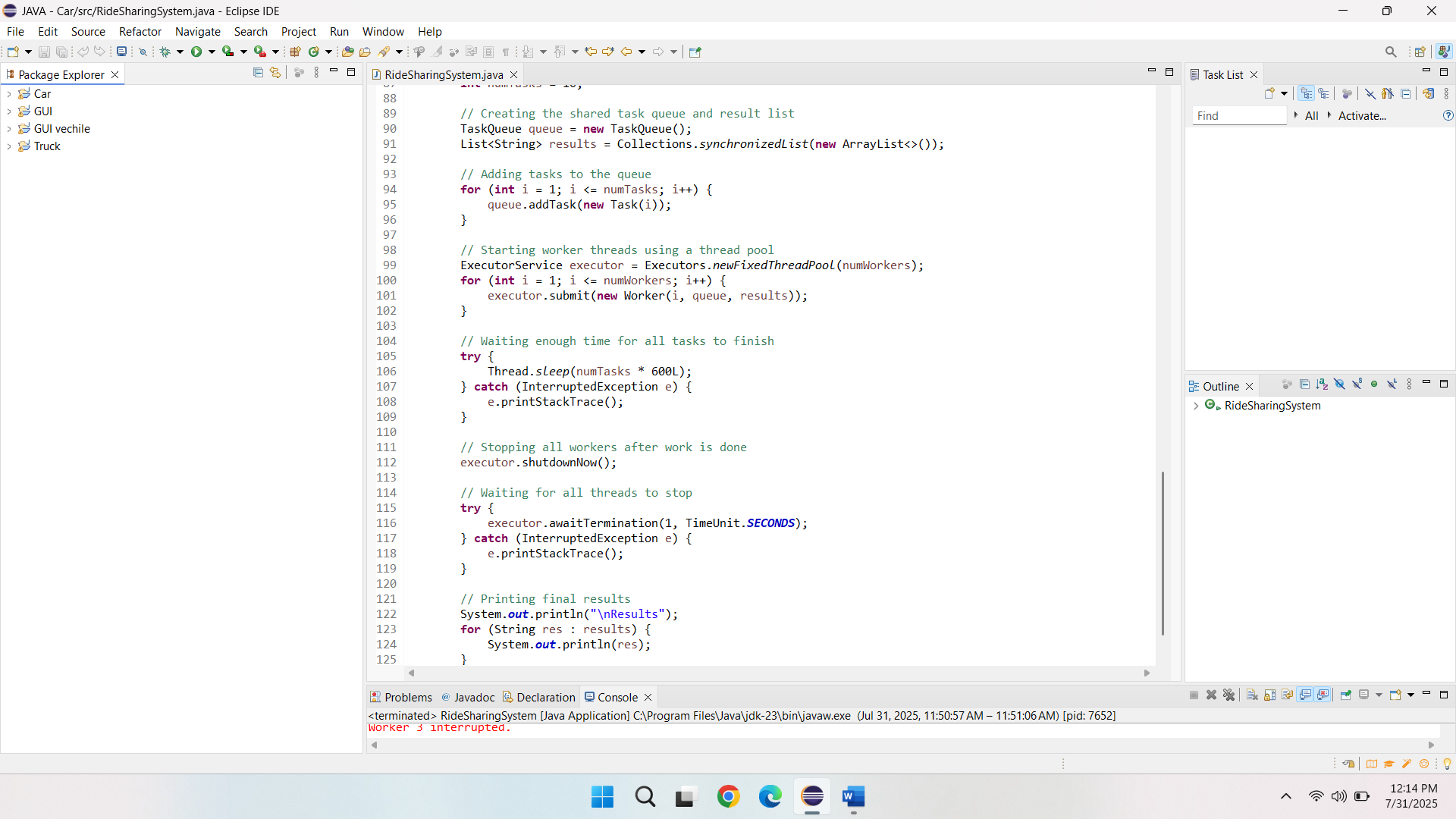
The Go version embraces the communicating sequential processes model, which is based on goroutines and channels as constructs to deliver lightweight and safe concurrency. The channel is a buffered channel, which is a queue of tasks being assigned safely and synchronised. Every goroutine picks up a task, does a delay to simulate processing, and posts the result to a distinct result channel. Wait Group is applied to monitor and block all goroutines until the end of their execution, after which the program exits. The error handling is taken care of by examining the values returned and making use of defer statements to clean up resources and log (Tipirneni, 2022). By increasing the number of channels, the need to explicitly lock and the coordination of the multiple processing processes becomes easier. Logging helps in tracing the beginning and end of the task and other processing errors, which aid in debugging and transparency.

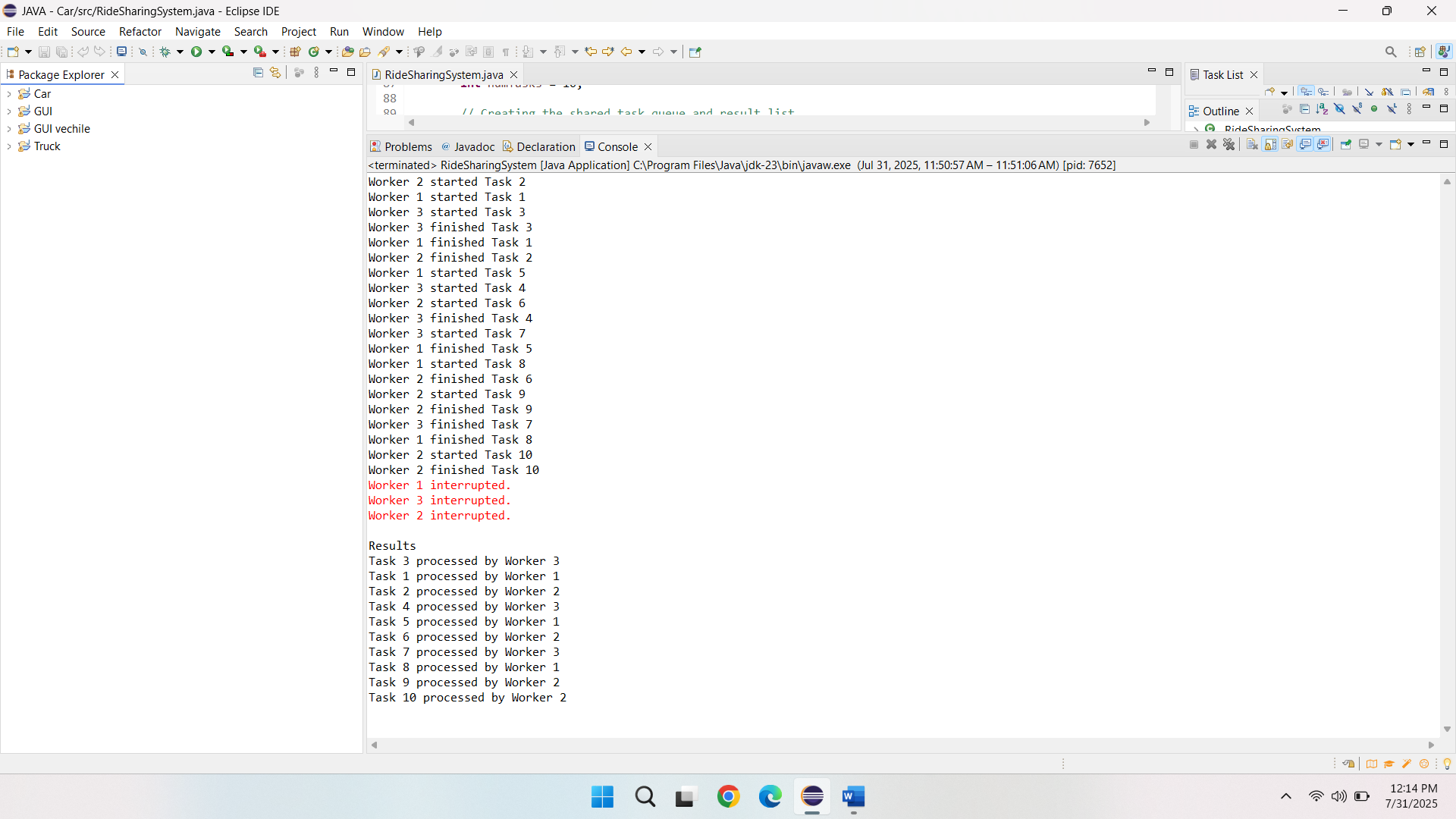
**Java Implementation**

The Java implementation starts with defining a task class and the task storage queue that will be thread-safe. A thread pool is created, and using it, a fixed number of worker threads are created and can process tasks concurrently. Once a thread gets the task, it simulates the process by using a delay and then writing the result to a global list. To prevent race conditions caused by updates, synchronization is used on the list of results. The primary approach bootstraps the system and introduces tasks and gets the workers off the ground, and manages exceptions so that the code executes smoothly.



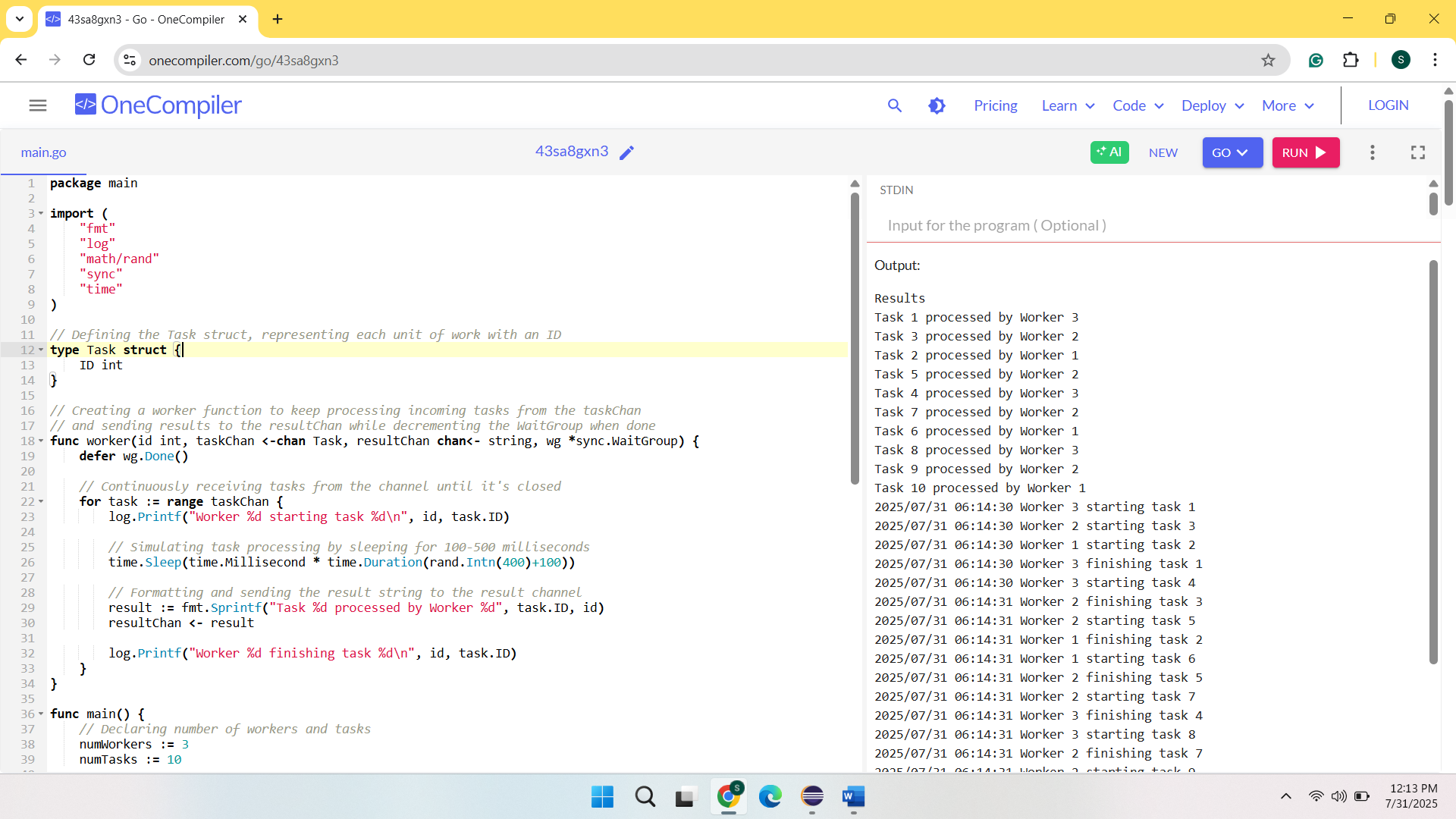


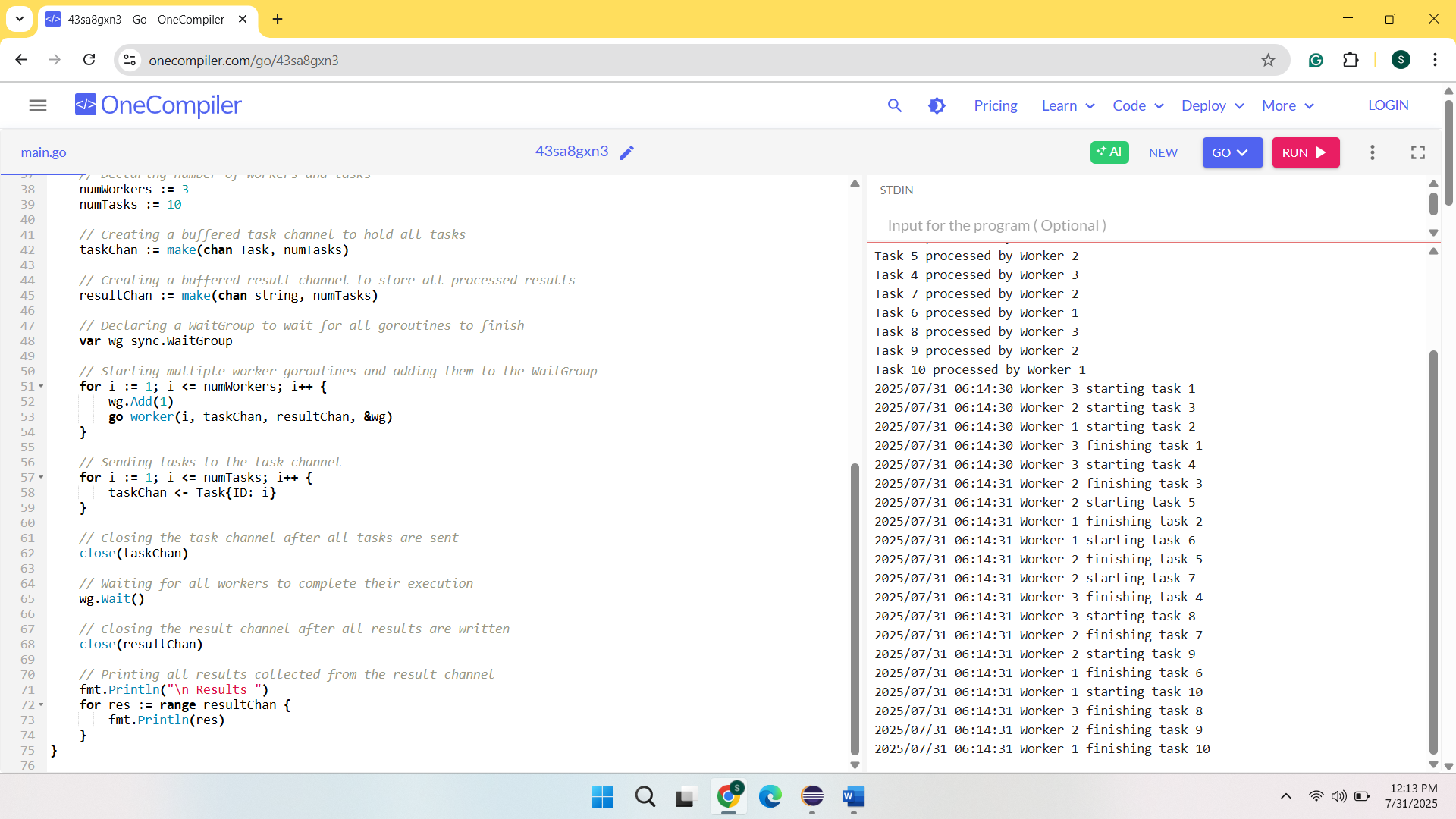




**GO Implementation**

Go has a task structure that is initialized in the implementation of Go and uses buffered channels in assigning tasks. To simulate processing, worker goroutines are created to get the tasks received over the channel and wait for a random period defined by a delay. Every result to be processed is then directed towards a result channel, and the Wait Group can monitor completion. Error handling is achieved with defer statements, and logs are used to trace the flow of tasks. When everything is done, the results are shown, and the system exits properly.





**Comparison of Concurrency Models in Java and Go**

Java and Go are two opposite versions of dealing with concurrency. The thread-based model of Java gives comprehensive control of the execution of threads and synchronization. Such a model necessitates the explicit shared resource management, adding complexity but enabling the optimization of performance to a fine-grained level. Conversely, Go follows a message-passing paradigm that neither includes shared memory nor has excessive synchronization overhead. Channels enable inter-glyon communication safely so that the likelihood of deadlocks and race conditions is decreased without explicit locks. Its concurrency model in Go is more concise and intuitive, especially in applications of a large scale and a high limit of concurrency. Java is a solid option when it comes to applications that require powerful thread handling and fine-grained synchronization. These two implementations serve the same purpose, which is presenting the project objectives, yet they vary largely in syntax, structure, and simplicity of their application.

**Conclusion**

The Ride Sharing System effectively illustrates a practical implementation of the principles of concurrent programming with two paradigms. Java gives a robust and environment-friendly tool for constructing multi-threaded systems, but with a lot of complexity in the use of explicit synchronization. Go simplifies concurrency through the native support of goroutines and channels, which makes the code cleaner and safer. These two implementations satisfy the functional requirements, such as queuing of the tasks, parallel processing, logging of the results, exception handling, and safe termination. The two languages can be compared in that they give value analysis on how concurrency can be achieved using the various programming constructs and help system architects to use these constructs when designing a parallel processing application.

**References**

Khanna, D., Sharma, S., & Purandare, R. (2023, December). Verifying Exception-Handling Code in Concurrent Libraries. In *2023 30th Asia-Pacific Software Engineering Conference (APSEC)* (pp. 405-414). IEEE.

Tipirneni, D. S. (2022). *An Empirical Study of Concurrent Feature Usage in Go* (Doctoral dissertation, East Carolina University).