MSCS-632-B01 Advanced Programming Languages

Report on Concurrency and Exception Handling in Java and Go

Amit Yadav

ayadav36753@ucumberlands.edu

Dr. Dax Bradley

University of the Cumberlands

Concurrency and Exception Handling in Java and Go

Concurrency is a fundamental aspect of modern software systems, enabling applications to perform multiple operations simultaneously. This report compares how two programming languages - Java and Go - implement concurrency and handle exceptions, particularly in the context of a multi-threaded data processing system.

Concurrency in Java

Java achieves concurrency using threads, either by extending the Thread class or implementing the Runnable interface. For this project, the ExecutorService from the java.util.concurrent package was used to manage a pool of worker threads. Shared resources, such as the task queue and result list, were guarded using ReentrantLock and Condition objects to avoid race conditions and ensure safe thread access. These locks allowed workers to wait for tasks using await() and resume when tasks became available using signal().

Java's concurrency model is preemptive and low-level, requiring explicit thread management and synchronization. While this provides fine-grained control, it also increases the complexity of handling shared state and avoiding deadlocks.

Concurrency in Go

Go uses a different model known as communicating sequential processes (CSP). Instead of threads, Go uses goroutines, which are lightweight functions that run concurrently. Goroutines are managed by the Go runtime, making them more memory-efficient than Java threads. To

coordinate goroutines and share data safely, Go uses channels, which act as concurrency-safe queues.

In the Go implementation, worker goroutines pull tasks from a channel and send results to another channel. The sync.WaitGroup was used to ensure the main function waited for all workers to complete before writing the output to a file. This model minimizes the need for locks and makes concurrent code easier to reason about.

Exception and Error Handling

Java handles exceptions using try-catch blocks. Specific exceptions such as InterruptedException and IOException were caught to manage errors gracefully, including thread interruptions and file I/O failures. The use of structured exception handling ensures that resources like file writers are closed properly, even when errors occur.

Go, on the other hand, follows an explicit error return model. Most functions return an error as a second return value, which must be checked manually. For example, file creation and write operations check for errors and handle them using conditional logic. Additionally, Go's defer keyword ensures that cleanup actions (such as closing files) are executed regardless of whether an error occurred.

Comparison of Concurrency Models

Java's thread-based concurrency model offers extensive control but requires careful synchronization to avoid race conditions and deadlocks. In contrast, Go's goroutine and channel model abstracts much of this complexity, promoting safer and more intuitive concurrent code. Java's exception handling is centralized and automatic, while Go's error handling is manual and explicit, emphasizing developer responsibility.

References

Go Programming Language. (n.d.). Effective Go: Concurrency. Retrieved from https://go.dev/doc/effective_go#concurrency

Oracle. (n.d.). The Java™ Tutorials: Concurrency. Retrieved from https://docs.oracle.com/javase/tutorial/essential/concurrency/