Read-Write Lock

A Read-Write Lock in Java is a concurrency control mechanism that allows multiple threads to read shared data simultaneously, but only one thread to write to it at a time. This improves performance in cases where reading happens more frequently than writing, since multiple readers can access the resource concurrently without blocking each other. Java provides the ReentrantReadWriteLock class from the java.util.concurrent.locks package to implement this lock.

Key Points:

* Multiple Readers, Single Writer: Multiple threads can acquire the read lock concurrently, as long as no thread holds the write lock. However, if a thread acquires the write lock, no other thread (reader or writer) can acquire any lock until the write lock is released.
* Read Lock: Allows multiple threads to read the shared resource as long as no thread holds the write lock. This improves throughput in read-heavy applications.
* Write Lock: Only one thread can acquire the write lock at a time. It ensures exclusive access to the shared resource for modifications.
* Lock Downgrade: The lock can be "downgraded" from a write lock to a read lock, meaning a thread holding the write lock can acquire a read lock without releasing the write lock first. But the reverse (upgrading from read to write) is not possible without releasing the read lock first.

Advantages:

* Improved performance: Especially in read-heavy scenarios, as multiple reads can happen concurrently without blocking each other.
* Flexibility: You can control when to lock for reading or writing based on the situation.

Disadvantages:

* Complexity: Managing read-write locks can be more complex compared to a regular lock.
* Writer Starvation: In some cases, if the read lock is acquired too frequently, writers may be starved, i.e., they may have to wait indefinitely.

Use Cases:

* Systems where there is a lot of reading and relatively less writing (e.g., caching systems).

Example