perplexity

Explain the ReentrantLock class and its features compared to using synchronized blocks

`ReentrantLock` (from `java.util.concurrent.locks`) is a modern alternative to the `synchronized` keyword. While both provide **mutual exclusion** (only one thread can enter the critical section) and reentrancy (a thread can re-acquire a lock it already holds), `ReentrantLock` offers much more flexibility and control.[1] [2]

In a nutshell: `synchronized` is easy but rigid; `ReentrantLock` is powerful but requires careful manual coding (`lock()` and `unlock()`).[3]

Key Features (Why use ReentrantLock?)

1. Try-Lock (Non-blocking Attempt)

- **Synchronized:** A thread trying to enter a `synchronized` block **must block** indefinitely if the lock is held by another thread. It cannot "give up" or do something else.
- **ReentrantLock:** You can use `tryLock()`. It attempts to acquire the lock immediately. If successful, it returns `true`. If the lock is busy, it returns `false` immediately (or after a timeout), allowing the thread to do other work instead of waiting forever.[4] [5]
    - *Code:* `if (lock.tryLock(1, TimeUnit.SECONDS)) { ... }`

2. Interruptible Locking

- **Synchronized:** A thread waiting for a synchronized lock **cannot be interrupted**. You cannot "cancel" the wait.
- **ReentrantLock:** You can use `lockInterruptibly()`. If a thread is waiting for the lock, you can call `thread.interrupt()` to wake it up and stop it from waiting. This is crucial for responsive applications.[2] [5]

3. Fairness (FIFO)

- **Synchronized:** Is **unfair**. If 5 threads are waiting, and the lock becomes free, *any* thread (even a new one that just arrived) might grab it. This can lead to **thread starvation**.
- **ReentrantLock:** You can create it with fairness enabled: `new ReentrantLock(true)`. This guarantees that the **longest-waiting thread** gets the lock next (First-In-First-Out).[6] [2]

4. Locking Across Methods

- **Synchronized:** Must start and end in the **same block/method**.

- **ReentrantLock:** You can call `lock()` in `methodA()` and `unlock()` in `methodB()`. This allows complex locking chains (e.g., Hand-over-hand locking in linked lists) that `synchronized` cannot handle.[5] [3]

Comparison Summary for Interview

Feature	`synchronized` keyword	`ReentrantLock`
Usage	Implicit (JVM handles lock/unlock)	Explicit (Must call `lock()` / `finally { unlock() }`)
Non-blocking?	No (Waits forever)	Yes (`tryLock()`)
Interruptible?	No	Yes (`lockInterruptibly()`)
Fairness?	No (Unfair)	Optional (Can be fair)
Performance	Better for low contention (JVM optimizations)	Better for high contention
Conditions	Only one `wait()`/`notify()` set	Multiple `Condition` objects (await/signal)

Best Practice Hint

"I default to `synchronized` because it's simpler and less error-prone (you can't forget to unlock). I only switch to `ReentrantLock` if I specifically need **fairness**, **timeouts**, or **interruptibility**."[2]

⁂

1. https://www.tutorialspoint.com/difference-between-reentrantlock-and-synchronized-in-java
2. https://javanexus.com/blog/reentrantlock-vs-synchronized-tool
3. https://javarevisited.substack.com/p/difference-between-reentrantlock
4. https://www.youtube.com/watch?v=wTMHRf-qdPM
5. https://javarevisited.blogspot.com/2013/03/reentrantlock-example-in-java-synchronized-difference-vs-lock.html
6. https://ihelio.today/posts/2025/java-concurrency-revisited-synchronized-vs-reentrantlock/
7. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/429036f4-3bc3-4a7a-8c6f-90246e73b9b9/image.jpg
8. https://www.geeksforgeeks.org/java/reentrant-lock-in-java/
9. https://www.youtube.com/watch?v=_xnoK4Cynr0
10. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/24f18c0a-b56e-4862-a619-e2959036a5c2/image.jpg
11. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/97e9ed86-ed11-4035-a18b-1eb05c1bab4e/image.jpg
12. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/87f62423-96f2-4071-9802-8f6699e0ecd8/image.jpg

13. https://stackoverflow.com/questions/11821801/why-use-a-reentrantlock-if-one-can-use-synchronizedthis
14. https://www.reddit.com/r/java/comments/1gmeeny/comparison_of_synchronized_and_reentrantlock/