**1 ReentrantLock basics**

The ReentrantLock class is part of the java.util.concurrent.locks package. It implements the Lock interface, which defines the basic operations of a lock: lock, unlock, tryLock, and lockInterruptibly. A ReentrantLock object has two main characteristics: it is reentrant and it is fair. Reentrant means that a thread can acquire the same lock multiple times without blocking itself. Fair means that the lock grants access to the longest-waiting thread, rather than the one that arrives first.

**2 ReentrantLock vs synchronized**

The synchronized keyword is another way to achieve thread synchronization in Java, which can be applied to a method or a block of code and uses an implicit lock associated with the object or the class. Compared to the ReentrantLock, the synchronized keyword is more restrictive since it automatically locks and unlocks at the beginning and end of the scope, while the ReentrantLock allows you to lock and unlock at any point in your code. Additionally, the ReentrantLock provides more features such as interrupting a waiting thread, trying to acquire a lock for a specified time, and checking if the lock is held by the current thread or any thread. Furthermore, the ReentrantLock can be fair or unfair, whereas the synchronized keyword is always unfair.

**🔑 Original Line:**

“**The ReentrantLock provides more features such as interrupting a waiting thread, trying to acquire a lock for a specified time, and checking if the lock is held by the current thread or any thread.**”

**💡 What is ReentrantLock?**

It’s a **Java class used for thread synchronization**, just like synchronized.  
But it gives **more control and flexibility** compared to synchronized.

**🔍 Now Let’s Break the Line:**

**✅ 1. Interrupting a waiting thread**

* When a thread is waiting to acquire a lock, **you can interrupt it** and make it stop waiting.
* This is **not possible** with the synchronized block.

**Example:**

ReentrantLock lock = new ReentrantLock();

Thread t1 = new Thread(() -> {

try {

lock.lockInterruptibly(); // This allows interruption

// Do work

} catch (InterruptedException e) {

System.out.println("Thread was interrupted while waiting");

}

});

**✅ 2. Trying to acquire a lock for a specific time**

* Instead of waiting forever, you can tell the thread:  
  **"Try to get the lock, but only wait for X seconds."**
* After that, it will **stop waiting** even if it didn't get the lock.

**Example:**

if (lock.tryLock(2, TimeUnit.SECONDS)) {

// Got the lock

} else {

// Didn't get the lock in time

}

**✅ 3. Checking if the lock is held**

* You can check:
  + lock.isHeldByCurrentThread() → whether **this thread** holds the lock
  + lock.isLocked() → whether **any thread** holds the lock

**Example:**

if (lock.isHeldByCurrentThread()) {

System.out.println("I have the lock");

}

if (lock.isLocked()) {

System.out.println("Someone has the lock");

}

**🔚 In Simple Words:**

ReentrantLock is like a smarter version of synchronized that lets you:

* 💥 Cancel waiting if needed
* ⏰ Wait for limited time
* 👁️ Check lock status anytime

Let me know if you want a **real-world example** using ReentrantLock in a mini project or multithreading question for interviews.