**What is Reentrant Lock in Java?**

A Reentrant Lock in Java is an advanced tool for synchronization, which, like the synchronized block or method, allows threads to gain control over a resource but with more flexibility and extended capabilities. It is part of the java.util.concurrent.locks package, introduced in Java 5 to address some of the limitations of the traditional synchronized approach. Here are key features and aspects of Reentrant Locks:

**Reentrancy:** The term "reentrant" means that locks can be acquired multiple times by the same thread without causing the thread to block. This is particularly useful when a synchronized method or block needs to call another synchronized method on the same object, allowing the thread to acquire the lock again without deadlocking. The lock keeps a hold count to ensure that the thread must release the lock as many times as it has acquired it.

**Lock Acquisition and Release**

Unlike the implicit locking mechanism provided by the synchronized keyword, which acquires and releases a lock implicitly when entering and exiting a synchronized block or method, Reentrant Locks offer explicit lock acquisition and release. This is done through its lock() and unlock() methods, providing more control over the lock.

ReentrantLock lock = new ReentrantLock();

lock.lock();

try {

// access the resource protected by this lock

} finally {

lock.unlock(); // ensure the lock is unlocked even if an exception is thrown

}

**Fairness Policy**

Reentrant Locks can be created with a fairness policy. A fair lock is one where threads acquire the lock in the order they asked for it (FIFO), which can reduce thread starvation but may decrease throughput compared to an unfair lock.

**Condition Support**

Reentrant Locks also support one or more Condition objects, which allow threads to wait for certain conditions to be true within a locked block, similar to the way Object.wait() and Object.notify() work in synchronized blocks/methods but with more flexibility. Conditions can help coordinate the activities of multiple threads by allowing them to wait for specific conditions to occur.

**Comparison with Synchronized**

Flexibility and Control: Reentrant Locks provide methods for timed, polled, or interruptible lock acquisitions, offering more control than synchronized blocks.

Fairness: The ability to specify a fairness policy is something synchronized blocks do not support.

Condition Variables: They allow splitting synchronized conditions into multiple sets for more fine-grained control, unlike the single wait-set available per object monitor in traditional synchronization.

Reentrant Locks are powerful tools for situations where more control over locking is needed, but they also require more careful management to avoid deadlocks and ensure that locks are always properly released. The choice between using synchronized blocks/methods and Reentrant Locks should be based on specific requirements for flexibility, control, and performance.