**Fairness of Locks** refers to the concept of ensuring that threads acquire a lock in a fair and predictable order. In Java, fairness primarily comes into play with certain types of locks, especially with ReentrantLock, which offers a *fair mode*. Here’s a breakdown of fairness in locks:

**1. Fair vs. Non-Fair Locks**

* **Fair Lock**: Guarantees that threads waiting for the lock are granted access in the order they requested it (FIFO order).
  + With a fair lock, no thread "jumps the queue"—the longest-waiting thread is the next to acquire the lock.
  + This approach prevents *starvation* (where some threads never get access), but it can result in lower throughput because the scheduler is forced to follow the queue order strictly.
* **Non-Fair Lock**: Allows threads to "jump the queue" and acquire the lock, potentially leading to higher throughput.
  + Non-fair locks do not guarantee any specific order, so any thread may acquire the lock whenever it becomes available.
  + Although non-fair locks improve performance by making the lock available to any thread as soon as it's free, they can lead to starvation for some threads.

**2. Implementation in Java (ReentrantLock)**

* ReentrantLock in Java allows for fair or non-fair lock acquisition.
  + When created as ReentrantLock(true), the lock is fair, ensuring threads acquire the lock in the order they requested it.
  + When created as ReentrantLock(false) or by default, it is non-fair, which can increase efficiency but does not guarantee order.
* **Example of Fair Lock Creation**:

// Creating a fair ReentrantLock

private final ReentrantLock lock = new ReentrantLock(true);

public void accessResource() {

lock.lock();

try {

// critical section code

} finally {

lock.unlock();

}

}

This ensures that threads waiting to enter the critical section will do so in the order they arrived.

**Summary**

Choosing between a fair or non-fair lock depends on the specific requirements of your application:

* **Fair Lock**: Better for predictable access and prevents starvation but at the cost of some performance.
* **Non-Fair Lock**: Higher throughput and lower latency, suitable for performance-critical applications where strict order is not essential.

Let me know if you'd like more on specific lock usage in an application context!