Yes bhai, you're right — in both wait/notify and ReentrantLock/Condition, we **call methods on the same object** (like store.produce() and store.consume()). But the **key difference is not the object**, it's **how we manage the waiting and signaling inside**.

Let me show you exactly what’s going on 👇

**✅ 1. In wait()/notify() (synchronized)**

public synchronized void produce() {

while (buffer full) {

wait(); // waits on 'this'

}

add item

notify(); // notifies someone waiting on 'this'

}

🔁 **Producer and Consumer both wait on the same monitor: this**

* Producer might notify another **producer**
* Consumer might notify another **consumer**

That can cause **no one to proceed** (deadlock/hang).

**✅ 2. In ReentrantLock + Condition**

ReentrantLock lock = new ReentrantLock();

Condition notFull = lock.newCondition();

Condition notEmpty = lock.newCondition();

public void produce() {

lock.lock();

try {

while (buffer full) {

notFull.await(); // only producers wait here

}

add item

notEmpty.signal(); // notify waiting consumers

} finally {

lock.unlock();

}

}

public void consume() {

lock.lock();

try {

while (buffer empty) {

notEmpty.await(); // only consumers wait here

}

remove item

notFull.signal(); // notify waiting producers

} finally {

lock.unlock();

}

}

✅ **Now, Producer waits on notFull, Consumer waits on notEmpty**  
🔁 And we notify **only the correct type**.

**💡 So What's the Difference?**

Even though store.produce() and store.consume() are called on the **same object**, the **waiting mechanism is separated internally** using:

| **Mechanism** | **wait/notify** | **Condition** |
| --- | --- | --- |
| Wait location | All threads wait on this | Producers on notFull, consumers on notEmpty |
| Risk of wrong wakeup | ✅ Yes | ❌ No |
| Wake control | Only notify() or notifyAll() | signal() to specific condition |
| Thread grouping | ❌ Same pool | ✅ Separated pools |

**🔧 Analogy**

Imagine a classroom:

* wait()/notify() = all students (producers + consumers) sit in the **same room**. Teacher says “Wake up 1 student” → might wake up the wrong one 😬
* Condition = students are in **separate rooms**. Teacher says “Wake up 1 student from Consumer Room” → correct one wakes up 👍

**So even if you use the same object (store), the real advantage is:**

✅ **Multiple conditions inside the object**, not just the object itself.

Want me to give you a full working example of this? Or show how it avoids deadlock when multiple producers/consumers are involved?

Let’s go step-by-step 💪