You're asking about how to **synchronize a non-synchronized list** (like ArrayList) in Java — to make it **thread-safe** for use in **multi-threaded environments**.

Let’s break it down 🔧

**🟡 Problem: ArrayList is not synchronized**

So if multiple threads access and modify it at the same time, you may get:

* ConcurrentModificationException
* Data inconsistency
* Race conditions

**✅ Solution: Ways to Synchronize a List**

**1. Collections.synchronizedList()**

java

CopyEdit

import java.util.\*;

List<String> list = new ArrayList<>();

List<String> syncList = Collections.synchronizedList(list);

🔐 Now syncList is thread-safe.

💡 **Important**: If you're **iterating**, do it inside a synchronized block:

java

CopyEdit

synchronized(syncList) {

for (String item : syncList) {

System.out.println(item);

}

}

**2. Use CopyOnWriteArrayList (from java.util.concurrent)**

java

CopyEdit

import java.util.concurrent.CopyOnWriteArrayList;

List<String> threadSafeList = new CopyOnWriteArrayList<>();

✅ It's thread-safe and allows safe iteration **without needing synchronized blocks**.

📌 Best when reads are frequent, and writes are rare.

**3. Using synchronized block manually**

If you’re using a regular list:

java

CopyEdit

List<String> list = new ArrayList<>();

synchronized(list) {

list.add("Hello");

}

But this gets messy for large apps — better to use option 1 or 2.

**🔁 Summary**

| **Method** | **Thread-Safe** | **Best Use Case** |
| --- | --- | --- |
| Collections.synchronizedList | ✅ | Simple thread safety |
| CopyOnWriteArrayList | ✅ | Read-heavy, low-write apps |
| Manual synchronized block | ✅ | Fine-grained custom control |