Great follow-up, bro! Let’s clear this up. 🔥

**❓ Your Question:**

If the thread is **not sleeping**, what happens when we call interrupt()?

**✅ Answer:**

When a thread is **not in a blocking state** (like sleep(), wait(), or join()), calling interrupt() **does not throw an exception**.  
Instead, it simply **sets a flag** — called the **interrupted flag**.

Thread.currentThread().isInterrupted() // returns true if interrupt() was called

So, in your code:

while (!Thread.currentThread().isInterrupted()) {

// loop body

}

👉 If the thread is **doing normal work (not sleeping)** and you call .interrupt(), it won’t be disturbed immediately.

🔁 But the **next time it reaches the while check**, isInterrupted() returns true, and the loop **exits** cleanly.

**🧠 Real-life analogy:**

Imagine you're telling someone:

“Hey, next time you look at your watch, remember you're supposed to stop working.”

They won't stop right now. But once they **check the watch**, they’ll stop. That’s what isInterrupted() does.

**✅ Summary:**

| **State of Thread** | **Behavior when interrupt() is called** |
| --- | --- |
| 💤 Sleeping / waiting | Wakes up + throws InterruptedException |
| ✅ Active (normal code) | No exception, flag is set |
| 🔁 Loop checks flag | Thread sees it next time and exits |

Want to see a custom task that checks this flag every few milliseconds while doing some CPU work? Let me know, I’ll code it for you! 💻⚙️