Great question! Let’s break down the **difference between interrupt() and isInterrupted()** — one of the **most asked** and **confusing** topics in Java threading 🔥

**✅ 1. interrupt()**

* **What it does**:  
  It **sends an interrupt signal** to a thread. It does **not stop** the thread immediately.
* **Effect**:  
  It sets the **interrupted flag = true** for the target thread.
* **If thread is sleeping/waiting/joining** → it **throws** InterruptedException.
* **If thread is not blocking** → it just sets the flag.

Thread t = new Thread(() -> {

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

// running...

}

});

t.start();

t.interrupt(); // <-- sets the flag true

**✅ 2. isInterrupted()**

* **What it does**:  
  It checks whether the **interrupted flag is true** for the current thread.
* **It does NOT clear/reset the flag.**

Thread.currentThread().isInterrupted(); // true or false

**✅ 3. interrupted() (static method)**

⚠️ Notice: this is **static** method → Thread.interrupted()

* **What it does**:  
  Checks **and clears** the interrupted flag **of the current thread**.
* This is used when you want to **check & reset** the interrupt status in one go.

Thread.interrupted(); // returns true if interrupted AND resets the flag to false

**🧠 Summary Table:**

| **Method** | **Type** | **What it Does** | **Clears Flag?** |
| --- | --- | --- | --- |
| interrupt() | instance | Sets the interrupted flag to true | ❌ |
| isInterrupted() | instance | Checks if the thread was interrupted | ❌ |
| interrupted() | static | Checks & clears **current thread’s** interrupt status | ✅ |

**🔥 Quick Code Example:**

Thread t = new Thread(() -> {

System.out.println(Thread.interrupted()); // false

Thread.currentThread().interrupt(); // set flag

System.out.println(Thread.interrupted()); // true (also clears)

System.out.println(Thread.interrupted()); // false

});

t.start();

Want a practical example where we handle interruption for stopping a long-running task? I can show that too!