**✅ Your Setup Recap:**

* **Master**: Arduino #1 (connected to DHT22)
* **Slave**: Arduino #2 (receives data, prints on Serial, blinks LED)
* Communication: **I²C protocol** (SDA, SCL lines + common GND)

**🔍 Explanation According to the 9 Steps:**

**Step 1: Master sets SCL HIGH**

* On Arduino, the Wire.begin() library automatically manages the I²C bus.
* When idle, both **SCL** and **SDA** lines stay HIGH (due to pull-up resistors, typically 4.7kΩ to 5V).

**Step 2: Master pulls SDA LOW while SCL is HIGH (START Condition)**

* When your code calls:
* Wire.beginTransmission(8);

The library generates the **START condition**: SDA goes LOW while SCL is HIGH.

* This signals all devices on the bus that a message is about to start.

**Step 3: Master sends Slave Address (7-bit)**

* In your experiment, the slave address is 8.
* I²C uses **7 bits for address** + **1 bit for Read/Write**.
* Internally, the library sends:
* 00010000 (for write)

MSB first, synchronized with SCL clock pulses.

**Step 4: Master sends Read/Write Bit**

* Since Wire.beginTransmission(8) means **WRITE mode**, the R/W bit = 0.
* If you used Wire.requestFrom(8, n), then R/W bit = 1.

**Step 5: Master waits for ACK**

* After sending the address + R/W bit, the master releases SDA.
* The slave (Arduino #2) pulls SDA LOW during the ACK clock pulse to confirm:  
  ✔ "I am here and ready."

**Step 6: Slave sends ACK**

* Managed automatically by Arduino's hardware I²C interface.
* If the wrong address was sent, no ACK → Master knows the slave didn’t respond.

**Step 7: Data Transfer Begins**

* For each Wire.write() in the master code:
* Wire.write("DHT22 Data");

It sends **one byte at a time** on SDA while toggling SCL.

* Data is valid when SCL is HIGH.
* In **Write mode**, the master sends bytes; in **Read mode**, the slave sends.

**Step 8: ACK after Each Byte**

* After every byte:
  + The slave ACKs (pulls SDA LOW during ACK clock pulse).
* This ensures no data is lost. If no ACK, the master stops transmission.

**Step 9: STOP Condition**

* After finishing data transfer:
* Wire.endTransmission();

The library generates **STOP condition**:

* + SDA goes HIGH while SCL is HIGH.
  + This releases the bus for other communications.

**Speed Modes in Context**

* Arduino defaults to **Standard Mode** (100 kbps).
* You can change speed:
* Wire.setClock(400000); // Fast mode (400 kbps)
* High-speed (3.4 Mbps) and Ultra Fast (5 Mbps) require advanced hardware, not typical Arduino.

✅ **How Your Experiment Fits These Steps**

* When Arduino #1 sends T:26.45 and H:52.30, it’s following steps **2 to 9** repeatedly.
* The slave blinks LED (acknowledging reception) and prints data via Serial.

🔥 **Debugging Tip**:  
If you want to *see these steps in action*, use a **logic analyzer** (e.g., Saleae Logic or a cheap USB analyzer) to capture SDA and SCL waveforms. You’ll see START, address, ACK, data bytes, and STOP clearly.

