Great follow-up! Let’s break down what **effectively final** means and **why final (or effectively final) is required** when using variables inside lambdas or anonymous inner classes in Java.

**🔹 What is *effectively final*?**

A variable is **effectively final** if it is **not explicitly marked as final**, but **its value never changes** after initialization.

For example:

int x = 10;

// x is not declared final, but since we don't change x later,

// it is \*effectively final\*

Runnable r = () -> System.out.println(x); // ✅ Works fine

But this is NOT effectively final:

int x = 10;

x = 20; // ❌ Now x is not effectively final

Runnable r = () -> System.out.println(x); // Compilation error

**🔸 Why does lambda need *final* or *effectively final*?**

Java **lambdas and anonymous inner classes** can only use **final or effectively final variables** from the surrounding method.

This rule exists because:

* Lambdas **do not capture the value**, they capture a **reference**.
* If the variable changes after being captured, the lambda might refer to an **unexpected value**.
* To avoid **data inconsistency and thread-safety issues**, Java enforces this.

**🔹 Back to your case: Why this?**

final int tasknumber = i;

You’re inside a loop. The variable i is changing in every iteration — so it's **not effectively final**.

To safely pass its value into the lambda, you **create a new variable tasknumber and make it final**, so each lambda captures a unique value.

✅ This is safe and allowed.

**💡 Real-life analogy:**

Imagine a guy writing tasks on sticky notes and passing them to threads.

* If he writes directly on 1 sticky note (i), and keeps overwriting it in each loop — all threads read the final overwritten value.
* But if he makes **a copy of each note** (final tasknumber = i), every thread gets its **own unique note** with the correct value.

**✅ Summary**

| **Term** | **Meaning** |
| --- | --- |
| final variable | Explicitly cannot be changed |
| **Effectively final** | Not declared final, but never reassigned |
| Lambdas & anonymous classes | Can only use final or effectively final variables from outer scope |

Want a code example that shows this with compile-time errors if final is not used?