Potential logical issues

1. **Empty string after deletions** ○ sb.charAt(0) in a do...while can throw exceptions if sb becomes empty.
2. **Performance** ○ Repeated calls to indexOf and lastIndexOf inside loops are **O(n²)** in worst case.
3. **Loop counters (i and count) are arbitrary** ○ You are iterating do...while (i != sb.length()). If multiple # exist, the loop may not fully process them all, or could do extra iterations.
4. **Nested # logic** ○ If you have strings like abc###, the deletion should remove c then b then
    a. Your code works, but the logic is convoluted.
5. **Consecutive leading #** ○ Your do...while removes them correctly, but the code is hard to read and depends on exceptions.
6. **Negative index risk** ○ sb.charAt(indexLastHash-1) can become negative if # is at index 0. You handle it in catch, but it's better to check before accessing.

3. Real-life / practical issues with this style of logic

Now, imagining this code in a real-life project or a bigger system, some issues show up—not because your idea is bad, but because the implementation is a bit fragile and complex.

a) Complexity & maintainability Right

now the logic is:

- Nested try/catch blocks.
- Multiple do...while loops.
- Conditions that depend on the *current* sb.length() while you're mutating sb.
- Many repeated calls like sb.indexOf("#") and sb.charAt(0) inside loops.

If someone else (or future-you) opens this code months later, they'll probably think:

"I don't want to touch this. I'm scared I'll break some edge case I don't understand."

In real-life code, **simplicity is a survival skill**. Even if this works for all your tests, it's hard to *prove* it's correct for every possible input, or to safely extend it. **b) Using exceptions for control flow**

You’re catching StringIndexOutOfBoundsException and then doing extra cleanup. That suggests:

- The normal control flow and edge-case handling aren’t fully trusted by you (understandable; the logic is tricky).
- The exception becomes “another branch” of logic.

In production code, exceptions are usually for “something went wrong that we didn’t expect”, not “just another path when indices get awkward”. Using exceptions like this can hide real bugs and make performance worse. **c) Performance / efficiency** You repeatedly:

- Search for # from the start and from the end (indexOf, lastIndexOf).
- Delete from the middle of the string (which shifts everything to the right).
- Loop based on a length that’s changing as you delete.

For typical input lengths this is fine, but **as a pattern**, it’s closer to $O(n^2)$ than $O(n)$. In a real-time system (say, processing logs, user input streams, or giant text files), this approach might become too slow.

d) Risk of hidden edge cases

Because the logic is spread across:

- Multiple passes,
- try/catch blocks,
- and mutation inside loops that use sb.length() as a boundary, it’s very easy

to have some *weird* string like:

- Mixed Unicode,
- Very long sequences of #,
- Or an empty string after deletions that triggers a path you didn’t think of. In

production, weird inputs always show up eventually.