The recursion that never unwinds

You still have this cycle:

- generate3x3Grids() **calls** fill9x9(mp) (at the end)
- fill9x9() **calls** sudokuComplete(...) (when totalNumbersProcessed % 81 == 0)
- sudokuComplete() **calls** generate3x3Grids()

So you get:

generate3x3Grids → fill9x9 → sudokuComplete → generate3x3Grids → fill9x9 → ...

Each "board attempt" adds another layer to the call stack and never returns → eventually StackOverflowError.

Why this version overflows even sooner

In *this* version, you changed realTime9x9Fill() to do:

checkUniqueRows(formattedBoard, rowIndex);

checkUniqueColumns(formattedBoard, colIndex);

And both checkUniqueRows() and checkUniqueColumns() can call:

sudokuComplete(failedRows, failedColumns);

That means you now also have an **early recursion path** that can trigger *during the filling of a single board* (after any number is placed):

fill9x9 → realTime9x9Fill → checkUniqueRows/Columns → sudokuComplete → generate3x3Grids → fill9x9 → ...

So recursion can happen many times before you even reach 81 cells, and it stacks up extremely quickly.

OLD CODE:

https://www.amitamlani.com/02112024/7/2/Main.java

That's a solid approach — your older version "works" because it's **single-pass and iterative**: it fills a board, calls sudokuComplete(), resets, and then the *same loop* naturally proceeds to the next attempt without **re-entering** fill9x9() / generate3x3Grids() recursively.

If you port your "violation memory" into the old design, keep these rules and you'll avoid stack issues *and* keep your new behaviour:

1) Don't call fill9x9() or generate3x3Grids() from inside sudokuComplete()

In the old code, sudokuComplete() just **reports + resets flags/board** and returns true/false. That's good. Keep it that way.

So:

- sudokuComplete() should **never** start a new board.
- The **outer loop** (the thing already looping board attempts) is what starts the next board.

2) Add violation-recording, but only as data collection (no control flow)

In checkUniqueRows/Columns():

- If you detect a duplicate, set failedRows/failedColumns = true
- And **record the prefix** of storeRetrieved3x3Grid into blockedPermutationNumberSequence[position]
- Don't call sudokuComplete() from there. Just record and let the fill continue.

That preserves "natural path out".

3) Apply the new "avoid known-bad prefix sequences" only when selecting the 9 permutation IDs

This is the key part you can drop into the old code cleanly:

- You already have the "pick 9 unique random indices" section.
- After you build storeRetrieved3x3Grid, check it against blockedPermutationNumberSequence.
- If it matches any blocked prefix, reject it and re-roll the 9 IDs.

So selection becomes:

- do { generate 9 unique permutation IDs } while (matchesAnyBlockedPrefix);

No recursion, no stack growth, and it’s easy to reason about.

MAIN STRUCTURAL FIX REQUIRED:

The simpler way, I had to ask ChatGPT several times since it mentioned techniques which were extremely out of my depth...

Do not call fill3x3() (or fill9x9()) from sudokuComplete() at all.
Instead, make fill9x9() run a loop that repeatedly attempts boards, and call sudokuComplete() just to **report success/failure**.
//This change was consistent with utilising my old code as a baseline

---

Minimal changes that fix it

1) Remove the recursion from sudokuComplete()

---

2) Put the "start next board" loop inside fill9x9()

Wrap your whole board-generation in a while loop, and when a board finishes, **break out of the nested loops** and let the while start a new board.

Here's the key pattern (drop-in style):

```
public void fill9x9(Map<Integer, int[][]> mp)
{
  while (numComplete9x9Boards.compareTo(numberPossibleBoards) < 0)
  {
    // reset per-board state here (NOT inside sudokuComplete)
  }
```

That's the "simple way back into fill9x9": **a loop**, not a method call.

One more bug you should fix now (or your selection logic will misbehave)

Summary

- Your StackOverflow happens because sudokuComplete() calls fill3x3() → recursion chain.
- The simplest correct design is: **fill9x9() loops boards**, and **sudokuComplete() only reports**.