THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."

HEY! GET BACK TO WORK!

COMPILING!

OH. CARRY ON.
Backtracking strategies

- When solving a backtracking problem, ask these questions:
  - What are the "choices" in this problem?
    - What is the "base case"? (How do I know when I'm out of choices?)
  - How do I "make" a choice?
    - Do I need to create additional variables to remember my choices?
    - Do I need to modify the values of existing variables?
  - How do I explore the rest of the choices?
    - Do I need to remove the made choice from the list of choices?
  - Once I'm done exploring, what should I do?
  - How do I "un-make" a choice?
The "8 Queens" problem

- Consider the problem of trying to place 8 queens on a chess board such that no queen can attack another queen.

- What are the "choices"?

- How do we "make" or "un-make" a choice?

- How do we know when to stop?
Naive algorithm

- for (each square on board):
  - Place a queen there.
  - Try to place the rest of the queens.
  - Un-place the queen.

- How large is the solution space for this algorithm?
  - $64 \times 63 \times 62 \times \ldots$
Better algorithm idea

- Observation: In a working solution, exactly 1 queen must appear in each row and in each column.

- Redefine a "choice" to be valid placement of a queen in a particular column.

- How large is the solution space now?
  - $8 \times 8 \times 8 \times ...$
Suppose we have a `Board` class with these methods:

<table>
<thead>
<tr>
<th>Method/Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public Board(int size)</code></td>
<td>construct empty board</td>
</tr>
<tr>
<td><code>public boolean isSafe(int row, int column)</code></td>
<td>true if queen can be safely placed here</td>
</tr>
<tr>
<td><code>public void place(int row, int column)</code></td>
<td>place queen here</td>
</tr>
<tr>
<td><code>public void remove(int row, int column)</code></td>
<td>remove queen from here</td>
</tr>
<tr>
<td><code>public String toString()</code></td>
<td>text display of board</td>
</tr>
</tbody>
</table>

Write a method `solveQueens` that accepts a `Board` as a parameter and tries to place 8 queens on it safely.
- Your method should stop exploring if it finds a solution.
Recall: Backtracking

A general pseudo-code algorithm for backtracking problems:

Explore(\text{choices}):

- if there are no more \text{choices} to make: stop.
- else, for each available choice C:
  - Choose C.
  - Explore the remaining \text{choices}.
  - Un-choose C, if necessary. (backtrack!)
Exercise solution

// Searches for a solution to the 8 queens problem
// with this board, reporting the first result found.
public static void solveQueens(Board board) {
    if (solveQueens(board, 1)) {
        System.out.println("One solution is as follows:");
        System.out.println(board);
    } else {
        System.out.println("No solution found.");
    }
}

...
public static boolean solveQueens(Board board, int col) {
    if (col > board.size()) {
        return true; // base case: all columns are placed
    } else {
        // recursive case: place a queen in this column
        for (int row = 1; row <= board.size(); row++) {
            if (board.isSafe(row, col)) {
                board.place(row, col); // choose
                if (explore(board, col + 1)) { // explore
                    return true; // solution found
                }
                b.remove(row, col); // un-choose
            }
        }
    }
    return false; // no solution found
}