Problem Statement

You are given an integer n.

👉 Generate all combinations of **well-formed parentheses** containing n pairs of (and).

Example

Input: n = 3

Output: ["((()))","(()())","(())()","()(())","()()()"]

Key Idea

- Each valid sequence has exactly n opening and n closing brackets.
- At any point:
  - You can **add (** if you still have some left (open < n).
  - You can **add)** only if it won't break validity (close < open).
- Use **backtracking** to build all valid strings.

Java Solution (Backtracking)

```
import java.util.*;

class Solution {
    public List<String> generateParenthesis(int n) {
        List<String> result = new ArrayList<>();
        backtrack(result, "", 0, 0, n);
        return result;
    }

    private void backtrack(List<String> result, String current, int open, int close, int n) {
        // base case: if we used all parentheses
        if (current.length() == 2 * n) {
            result.add(current);
            return;
        }
```

```
        // add '(' if possible
        if (open < n) {
            backtrack(result, current + "(", open + 1, close, n);
        }

        // add ')' if valid
        if (close < open) {
            backtrack(result, current + ")", open, close + 1, n);
        }
    }
}
```

Dry Run Example (n = 3)

We want strings of length 6 (2 * 3).

Step Tree (simplified):

1. Start: "" (open=0, close=0)
   → add (→ "("
2. "(" (open=1, close=0)
   → "((" → "(((" → "(()" → "(()())", etc.
   → "()" → "())" …
3. Whenever open < n, we add (.
   Whenever close < open, we add).
   Only paths that respect these rules survive.

Example Path: "((()))"

- "" → "(" → "((" → "((("
- add) → "(()" → "(())" → "((()))" ✅

Example Path: "()()()"

- "" → "(" → "()" → "()(" → "()()" → "()()(" → "()()()" ✅

All 5 valid answers are generated.

Learning Points

1. **Backtracking**: try both choices at each step (add (or)), but only valid ones.

2. **Constraints ensure correctness**:
    - open <= n
    - close <= open
3. Time complexity ≈ **O(2^n)** in worst case, but pruned by rules, output size = Catalan number.