1. Leetcode Problem no. 36 – Valid Sudoku

**CODE :**

public class validsudoku {  
  
 public static boolean isValidSudoku(char[][] board) {  
 // Check rows  
 for (int i = 0; i < 9; i++) {  
 boolean[] seen = new boolean[9];  
 for (int j = 0; j < 9; j++) {  
 char currentVal = board[i][j];  
 if (currentVal != '.') {  
 int index = currentVal - '1';  
 if (seen[index]) {  
 return false;  
 }  
 seen[index] = true;  
 }  
 }  
 }  
  
 // Check columns  
 for (int j = 0; j < 9; j++) {  
 boolean[] seen = new boolean[9];  
 for (int i = 0; i < 9; i++) {  
 char currentVal = board[i][j];  
 if (currentVal != '.') {  
 int index = currentVal - '1';  
 if (seen[index]) {  
 return false;  
 }  
 seen[index] = true;  
 }  
 }  
 }  
  
 // Check subgrids  
 for (int block = 0; block < 9; block++) {  
 boolean[] seen = new boolean[9];  
 int rowOffset = (block / 3) \* 3;  
 int colOffset = (block % 3) \* 3;  
 for (int i = 0; i < 3; i++) {  
 for (int j = 0; j < 3; j++) {  
 char currentVal = board[rowOffset + i][colOffset + j];  
 if (currentVal != '.') {  
 int index = currentVal - '1';  
 if (seen[index]) {  
 return false;  
 }  
 seen[index] = true;  
 }  
 }  
 }  
 }  
  
 return true;  
 }  
  
 public static void main(String[] args) {  
 char[][] board = {  
 {'5', '3', '.', '.', '7', '.', '.', '.', '.'},  
 {'6', '.', '.', '1', '9', '5', '.', '.', '.'},  
 {'.', '9', '8', '.', '.', '.', '.', '6', '.'},  
 {'8', '.', '.', '.', '6', '.', '.', '.', '3'},  
 {'4', '.', '.', '8', '.', '3', '.', '.', '1'},  
 {'7', '.', '.', '.', '2', '.', '.', '.', '6'},  
 {'.', '6', '.', '.', '.', '.', '2', '8', '.'},  
 {'.', '.', '.', '4', '1', '9', '.', '.', '5'},  
 {'.', '.', '.', '.', '8', '.', '.', '7', '9'}  
 };  
  
 boolean isValid = *isValidSudoku*(board);  
 System.*out*.println("Is the Sudoku board valid? True or False\n"+isValid);  
 }

}

**Output :**

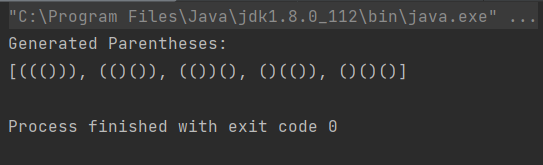
**A screen shot of a computer

Description automatically generated with medium confidence**

2. Leetcode Problem no. 22 – Generate Parentheses

**CODE :**

import java.util.ArrayList;  
import java.util.List;  
  
public class generateparentheses {  
  
 public static List<String> Parentheses(int n) {  
 List<String> result = new ArrayList<>();  
 StringBuilder sb = new StringBuilder();  
 *generateBrackets*(result, n, 0, 0, sb);  
 return result; }  
  
 private static void generateBrackets(List<String> result, int n, int opening, int closing, StringBuilder output){  
 // Termination Case  
 if(output.length() == n\*2){  
 result.add(output.toString());  
 return ; }  
 if(opening<n){  
 output.append("("); // Place Opening  
 *generateBrackets*(result, n, opening+1, closing, output);  
 output.deleteCharAt(output.length()-1); }  
 // Can i Place Closing  
 if(closing<opening){  
 output.append(")"); // Place Closing  
 *generateBrackets*(result, n, opening, closing+1, output);  
 // Stack Fall  
 output.deleteCharAt(output.length()-1);} }  
 public static void main(String[] args) {  
 int n = 3;  
  
 generateparentheses solution = new generateparentheses();  
 List<String> answer = solution.*Parentheses*(n);  
 System.*out*.println("Generated Parentheses:\n"+answer); }}

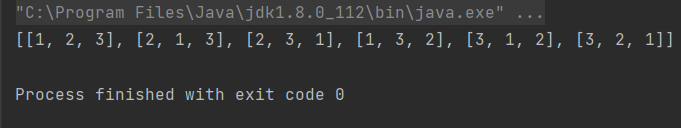
**Output :**

3. Leetcode Problem no. 46 – Permutations

**CODE :**

import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
  
public class Permutations {  
 public static List<List<Integer>> permute(int[] nums) {  
 List<List<Integer>> permutations = new ArrayList<>();  
 // Termination case  
 if (nums.length == 0) {  
 permutations.add(new ArrayList<>());  
 return permutations;  
 }  
 // Get the first element  
 int firstNum = nums[0];  
 // Generate permutations for the remaining elements  
 List<List<Integer>> result = *permute*(Arrays.*copyOfRange*(nums, 1, nums.length));  
  
 // Traverse the result  
 for (List<Integer> perm : result) {  
 for (int i = 0; i <= perm.size(); i++) {  
 List<Integer> newPerm = new ArrayList<>(perm);  
 newPerm.add(i, firstNum);  
 permutations.add(newPerm);  
 }  
 }  
 return permutations;  
 }  
  
 public static void main(String[] args) {  
 int[] nums = {1, 2, 3};  
 List<List<Integer>> result = *permute*(nums);  
 System.*out*.println(result);  
 }  
}

**Output :**

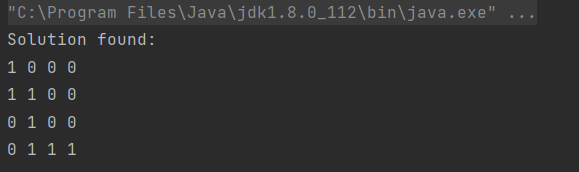
****

4. Rat in a Maze (Rat Move to Up , Left, Right, Down)

**CODE :**

public class RatMaze {  
 public static void main(String[] args) {  
 int[][] maze = {  
 {1, 0, 0, 0},  
 {1, 1, 0, 1},  
 {0, 1, 0, 0},  
 {1, 1, 1, 1}  
 };  
  
 int[][] solution = *solveMaze*(maze);  
  
 if (solution != null) {  
 System.*out*.println("Solution found:");  
 *printSolution*(solution);  
 } else {  
 System.*out*.println("No solution exists.");  
 }  
 }  
  
 public static int[][] solveMaze(int[][] maze) {  
 int rows = maze.length;  
 int cols = maze[0].length;  
 int[][] solution = new int[rows][cols];  
  
 if (*solveMazeUtil*(maze, 0, 0, solution)) {  
 return solution;  
 } else {  
 return null;  
 }  
 }  
  
 private static boolean solveMazeUtil(int[][] maze, int row, int col, int[][] solution) {  
 int rows = maze.length;  
 int cols = maze[0].length;  
  
 // Check if the current position is a valid move  
 if (row >= 0 && row < rows && col >= 0 && col < cols && maze[row][col] == 1) {  
 // Check if we have reached the destination  
 if (row == rows - 1 && col == cols - 1) {  
 solution[row][col] = 1;  
 return true;  
 }  
  
 // Mark the current position as part of the solution path  
 solution[row][col] = 1;  
  
 // Move to the right  
 if (*solveMazeUtil*(maze, row, col + 1, solution)) {  
 return true;  
 }  
  
 // Move down  
 if (*solveMazeUtil*(maze, row + 1, col, solution)) {  
 return true;  
 }  
  
 // Move to the left  
 if (*solveMazeUtil*(maze, row, col - 1, solution)) {  
 return true;  
 }  
  
 // Move up  
 if (*solveMazeUtil*(maze, row - 1, col, solution)) {  
 return true;  
 }  
  
 // If none of the moves lead to a solution, backtrack  
 solution[row][col] = 0;  
 return false;  
 }  
  
 return false;  
 }  
  
 public static void printSolution(int[][] solution) {  
 int cols = solution[0].length;  
  
 for (int[] ints : solution) {  
 for (int j = 0; j < cols; j++) {  
 System.*out*.print(ints[j] + " ");  
 }  
 System.*out*.println();  
 }  
 }  
 }

**Output :**

****

5. Leetcode Problem no. 79 – Word Search

**CODE :**

public class wordsearch {  
  
 public static boolean exist(char[][] board, String word) {  
 int rows = board.length;  
 int cols = board[0].length;  
  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < cols; j++) {  
 if (*dfs*(board, i, j, word, 0)) {  
 return true; } } }  
 return false; }  
  
 private static boolean dfs(char[][] board, int row, int col, String word, int index) {  
 if (index == word.length()) {  
 return true; }  
 int rows = board.length;  
 int cols = board[0].length;  
 if (row < 0 || row >= rows || col < 0 || col >= cols || board[row][col] != word.charAt(index)) {  
 return false; }  
 char temp = board[row][col];  
 board[row][col] = '#'; // Mark as visited  
 boolean found = *dfs*(board, row - 1, col, word, index + 1) ||  
 *dfs*(board, row + 1, col, word, index + 1) ||  
 *dfs*(board, row, col - 1, word, index + 1) ||  
 *dfs*(board, row, col + 1, word, index + 1);  
 board[row][col] = temp; // Restore the original character  
 return found; }  
  
 public static void main(String[] args) {  
 char[][] board = {{'A', 'B', 'C', 'E'}, {'S', 'F', 'C', 'S'}, {'A', 'D', 'E', 'E'} };  
  
 String word = "ABCCED";  
 boolean exists = *exist*(board, word);  
 System.*out*.println("Does the word exist? true or false\n"+exists);} }

**Output :**

