package com.iimtiaz.day\_18;  
  
public class ValidSudoku {  
 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'}};  
 System.*out*.println(new Solution().isValidSudoku(board));  
 }  
}  
  
*/\*\*  
Time Complexity: O(n^2), where n is the size of the Sudoku board (9 in this case).  
Nested Loops: The algorithm involves two nested loops that iterate through each cell of the 9x9 board, resulting in  
81 iterations.  
Constant-Time Operations: The operations within each iteration, such as accessing arrays, checking conditions, and  
marking numbers, are constant-time operations.  
Space Complexity: O(n^2)  
Three Boolean Arrays: The algorithm uses three boolean arrays (row, col, box) to track the presence of numbers in  
each row, column, and 3x3 box. Each array has a size of 9x9, leading to O(n^2) space usage.  
 \*/*class Solution {  
 public boolean isValidSudoku(char[][] board) {  
 boolean[][] row = new boolean[9][9];  
 boolean[][] col = new boolean[9][9];  
 boolean[][] box = new boolean[9][9];  
 for (int i = 0; i < 9; i++) {  
 for (int j = 0; j < 9; j++) {  
 if (board[i][j] != '.') {  
 int num = board[i][j] - '1';  
 int k = i / 3 \* 3 + j / 3;  
 if (row[i][num] || col[j][num] || box[k][num]) {  
 return false;  
 }  
 row[i][num] = col[j][num] = box[k][num] = true;  
 }  
 }  
 }  
 return true;  
 }  
}