سه تا متود داریم:  
solvesudoku, isSafe, print

تو متود اول: solvesudoku به صورت بازگشتی در ستون ها و سطرها میچرخیم، هروقت به سطر هشتم یا ستون نهم رسیدیم مقدار صحیح را برمیگردانیم تا متود متوقف شود و زمان سیو شود)این شرط خاتمه ی برنامه است). در ادامه ی کد هرجا خانه ی خالی دیدیم در حلقه ی for میاد از اعداد یک تا 9 رو میذاریم حای خونه ی خالی و میدیم به تابع isSafe. اگر در شرط سودوکو صدق میکرد عدد را در خانه ی خالی می نویسیم. اگه که صدق نمیکرد در خانه صفر میگذاریم و دوباره حلقه را ادامه می دهیم.

حالا متود isSafe : سه مقدار دریافت میکند: عددی که در خانه ی مورد نظر است ستون و سطری که در ان هستیم و عددی که برنامه قصد دارد در خانه قرار دهد.

در متود ابتدا یک حلقه ی for داریم که چک میکند اگر مقداری که قصد داریم بگذاریم در همان سطر موجود است false برمیگرداند در حلقه ی for بعدی هم همین را برای ستون چک می کند. و بعد هم خانه های 3\*3 را چک می کنیم.

Time complexity: O(pow(9,n\*n))

code:

import java.util.Scanner;  
  
// Java program for above approach  
public class sudokuClass {  
  
 // N is the size of the 2D matrix N\*N  
 static int *N* = 9;  
  
 /\* Takes a partially filled-in grid and attempts  
 to assign values to all unassigned locations in  
 such a way to meet the requirements for  
 Sudoku solution (non-duplication across rows,  
 columns, and boxes) \*/  
 static boolean solveSudoku(int grid[][], int row,  
 int col)  
 {  
  
 /\*if we have reached the 8th  
 row and 9th column (0  
 indexed matrix) ,  
 we are returning true to avoid further  
 backtracking \*/  
 if (row == *N* - 1 && col == *N*)  
 return true;  
  
 // Check if column value becomes 9 ,  
 // we move to next row  
 // and column start from 0  
 if (col == *N*) {  
 row++;  
 col = 0;  
 }  
  
 // Check if the current position  
 // of the grid already  
 // contains value >0, we iterate  
 // for next column  
 if (grid[row][col] != 0)  
 return *solveSudoku*(grid, row, col + 1);  
  
 for (int num = 1; num < 10; num++) {  
  
 // Check if it is safe to place  
 // the num (1-9) in the  
 // given row ,col ->we move to next column  
 if (*isSafe*(grid, row, col, num)) {  
  
 /\* assigning the num in the current  
 (row,col) position of the grid and  
 assuming our assigned num in the position  
 is correct \*/  
 grid[row][col] = num;  
  
 // Checking for next  
 // possibility with next column  
 if (*solveSudoku*(grid, row, col + 1))  
 return true;  
 }  
 /\* removing the assigned num , since our  
 assumption was wrong , and we go for next  
 assumption with diff num value \*/  
 grid[row][col] = 0;  
 }  
 return false;  
 }  
  
 /\* A utility function to print grid \*/  
 static void print(int[][] grid)  
 {  
 for (int i = 0; i < *N*; i++) {  
 for (int j = 0; j < *N*; j++)  
 System.*out*.print(grid[i][j] + " ");  
 System.*out*.println();  
 }  
 }  
  
 // Check whether it will be legal  
 // to assign num to the  
 // given row, col  
 static boolean isSafe(int[][] grid, int row, int col,  
 int num)  
 {  
  
 // Check if we find the same num  
 // in the similar row , we  
 // return false  
 for (int x = 0; x <= 8; x++)  
 if (grid[row][x] == num)  
 return false;  
  
 // Check if we find the same num  
 // in the similar column ,  
 // we return false  
 for (int x = 0; x <= 8; x++)  
 if (grid[x][col] == num)  
 return false;  
  
 // Check if we find the same num  
 // in the particular 3\*3  
 // matrix, we return false  
 int startRow = row - row % 3, startCol  
 = col - col % 3;  
 for (int i = 0; i < 3; i++)  
 for (int j = 0; j < 3; j++)  
 if (grid[i + startRow][j + startCol] == num)  
 return false;  
  
 return true;  
 }  
  
 // Driver Code  
 public static void main(String[] args)  
 {  
 int[][] grid = new int[9][9];  
 Scanner scan = new Scanner(System.*in*);  
 for(int i = 0; i < 9; i++) {  
 for (int j = 0; j < 9; j++) {  
 grid[i][j] = scan.nextInt();  
 }  
 }  
 if (*solveSudoku*(grid, 0, 0))  
 *print*(grid);  
 else  
 System.*out*.println("No Solution exists");  
 }  
}