public class SudokuGrid

```java
public class SudokuGrid {

    public final int SIZE = 9;
    public final int DIGIT_RANGE[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    public int SUDO[][] = new int[SIZE][SIZE];

    public SudokuGrid() {
        int[][] SU = {
            {1, 9, 5, 7, 0, 8, 4, 6, 2},
            {2, 6, 8, 4, 0, 9, 5, 0, 3},
            {3, 0, 4, 0, 2, 6, 9, 1, 0},
            {7, 5, 1, 3, 8, 2, 0, 4, 9},
            {4, 0, 9, 6, 5, 1, 2, 8, 7},
            {8, 2, 0, 9, 0, 7, 3, 0, 0},
            {0, 8, 2, 1, 9, 4, 7, 0, 0},
            {9, 0, 0, 8, 6, 3, 1, 2, 5},
            {6, 1, 3, 2, 0, 5, 0, 9, 0}};
        this.SUDO = SU;
    }
```

class SudokuSolver

```java
package sudoku;

import java.awt.Point;

public class SudokuSolver {

    public SudokuGrid grid;
    public int row = 0, col = 0;

    public boolean solve(SudokuGrid grid) {...31 lines }
}
```

void main

```java
package sudoku;

import java.awt.Point;
//import java.util.Scanner;

public class Sudoku {

    public static void main(String[] args) {

        SudokuGrid sudo = new SudokuGrid();

        SudokuSolver solver = new SudokuSolver();

        System.out.println(solver.solve(sudo));

        sudo.print();

    }

}
```

boolean solve

```java
public boolean solve(SudokuGrid grid) {

    Point point = new Point(row, col);

    if (grid.findEmptyCell(point)[0] != -1 || grid.findEmptyCell(point)[1] != -1) {

        System.out.print("row: " + grid.findEmptyCell(point)[0] + " col:" + grid.findEmptyCell(point)[1] +"  "+

        grid.fillCell(grid.findEmptyCell(point)[0], grid.findEmptyCell(point)[1]);

        if (col == 9) {
            col = 0;
            row++;
            if (row == 9)   return true;
            return solve(grid);
        }
        else if (row == 9)  return true;
        else   col++;

    } else {
        col++;
        if (col == 9) {
            col = 0;
            row++;
            if (row == 9)   return true;
            return solve(grid);
        } else if (row == 9)    return true;
    }
    return solve(grid);
}
```

public void fillCell

```java
public void fillCell(int r, int c) {
    //Scanner s = new Scanner(System.in);
    // a=s.nextInt();
    //Random random = new Random();
    int a = 0; // random.nextInt(9) + 1;
    while (a > 9 || a < 0 || givesConfict(r, c, a)) {
        if (a > 9 || a < 0) {
            System.out.println("Enter a namber from 1 to 9 .");
        } else {
            //System.out.println("This number is avaliable" + a);
        }
        a++;
        //a=random.nextInt(9) + 1;
        if (a == 9) {
            a = 0;
        }
    }

    if (!givesConfict(r, c, a)) {
        System.out.println("index=" + a);
        SUDO[r][c] = a;
    } else {
        System.out.println("false");
    }
}
```

boolean givesConfict

```java
public boolean givesConfict(int r, int c, int d) {

    int rowConflict[] = new int[9];
    int colConflict[] = new int[9];
    int boxConflict[][] = new int[3][3];

    for (int i = 0; i < SIZE; i++) {
        rowConflict[i] = SUDO[r][i];
        colConflict[i] = SUDO[i][c];
    }

    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {

            if (r < 3 && i < 3) {
                if (c < 3 && j < 3) {
                    boxConflict[i][j] = SUDO[i][j];
                } else if (c < 6 && c > 2 && j < 6 && j > 2) {
                    boxConflict[i][j - 3] = SUDO[i][j];
                } else if (c < SIZE && c > 5 && j < SIZE && j > 5) {
                    boxConflict[i][j - 6] = SUDO[i][j];
                }

            } else if (r < 6 && r > 2 && i > 2 && i < 6) {
                if (c < 3 && j < 3) {
                    boxConflict[i - 3][j] = SUDO[i][j];
                } else if (c < 6 && c > 2 && j < 6 && j > 2) {
                    boxConflict[i - 3][j - 3] = SUDO[i][j];
                } else if (c < SIZE && c > 5 && j < SIZE && j > 5) {
                    boxConflict[i - 3][j - 6] = SUDO[i][j];
                }

            } else if (r < SIZE && r > 5 && i > 5 && i < SIZE) {
                if (c < 3 && j < 3) {
                    boxConflict[i - 6][j] = SUDO[i][j];
                } else if (c < 6 && c > 2 && j < 6 && j > 2) {
                    boxConflict[i - 6][j - 3] = SUDO[i][j];
                } else if (c < SIZE && c > 5 && j < SIZE && j > 5) {
                    boxConflict[i - 6][j - 6] = SUDO[i][j];
                }

            }
        }
    }
    for (int i = 0; i < SIZE; i++) {

        if (rowConflict[i] == d || colConflict[i] == d) {
            return true;
        }
        if (i < 3) {
            for (int j = 0; j < 3; j++) {
                if (boxConflict[i][j] == d) {
                    return true;
                }
            }
        }
    }
    return false;
}
```

boş yerleri doldurmaya
onaylayıp doldurmak

```
row: 0 col:4   index=3
row: 1 col:4   index=1
row: 1 col:7   index=7
row: 2 col:1   index=7
row: 2 col:3   index=5
row: 2 col:8   index=8
row: 3 col:6   index=6
row: 4 col:1   index=3
row: 5 col:2   index=6
row: 5 col:4   index=4
row: 5 col:7   index=5
row: 5 col:8   index=1
row: 6 col:0   index=5
row: 6 col:7   index=3
row: 6 col:8   index=6
row: 7 col:1   index=4
row: 7 col:2   index=7
row: 8 col:4   index=7
row: 8 col:6   index=8
row: 8 col:8   index=4
```

sonuç

```
true
1 9 5 7 3 8 4 6 2
2 6 8 4 1 9 5 7 3
3 7 4 5 2 6 9 1 8
7 5 1 3 8 2 6 4 9
4 3 9 6 5 1 2 8 7
8 2 6 9 4 7 3 5 1
5 8 2 1 9 4 7 3 6
9 4 7 8 6 3 1 2 5
6 1 3 2 7 5 8 9 4
```

int[] findEmptyCell

boş yerleri bulmak

```java
public int[] findEmptyCell(Point point) {

    int empty[] = {-1, -1};
    if (point.x >= 9 || point.y >= 9) {
        System.out.println("point x=" + point.x + " point y=" + point.y);
        point.x=SIZE-1;
        point.y=SIZE-1;
    }
    if (SUDO[point.x][point.y] == 0) {
        empty[0] = point.x;
        empty[1] = point.y;
        return empty;
    }

    return empty;
}
```

TEST

```java
Point point;

for (int i = 0; i < 9; i++) {
    for (int j = 0; j < 9; j++) {

        point = new Point(i, j);

        if (sudo.findEmptyCell(point)[0] != -1) {
            System.out.println(sudo.findEmptyCell(point)[0] + " " + sudo.findEmptyCell(point)[1]);
        }
    }
}
```

Output - Sudoku (run)

```
run:
0 4
1 4
1 7
2 1
2 3
2 8
3 6
4 1
5 2
5 4
5 7
5 8
6 0
6 7
6 8
7 1
7 2
8 4
8 6
8 8
```