

```
1: package it.unisa.di.aa.sudoku;
2:
3: public class Coordinates {
4:     int r;
5:     int c;
6:
7:     public Coordinates(int r, int c) {
8:         this.r = r;
9:         this.c = c;
10:    }
11:
12:    public int getR() {
13:        return r;
14:    }
15:
16:    public int getC() {
17:        return c;
18:    }
19:
20: }
```

```
1: package it.unisa.di.aa.sudoku;
2:
3: class Solution implements Cloneable {
4:     int[][] numeri;
5:     int size;
6:     int sqrsz;
7:     int recursion_level;
8:
9:     public Solution(int[][] numeri, int size, int rl) {
10:         this.size = size;
11:         this.recursion_level = rl;
12:         sqrsz = (int) Math.sqrt(size);
13:         this.numeri = new int[size][size];
14:         for (int i=0; i<size; i++) {
15:             for (int j=0; j<size; j++) {
16:                 this.numeri[i][j]=numeri[i][j];
17:             }
18:         }
19:     }
20:
21:     public int[][] getNumeri() {
22:         return numeri;
23:     }
24:
25:     public int getRecursionLevel() {
26:         return recursion_level;
27:     }
28:
29:
30:     @Override
31:     protected Object clone() throws CloneNotSupportedException {
32:         return super.clone();
33:     }
34:
35:     public int getValue(int r, int c) {
36:         return numeri[r][c];
37:     }
38:
39:     public void setCell(int r, int c, int value) {
40:         System.out.println(String.format("Setting cell [%d][%d]=%d in solu
tion "+this,r,c,value));
41:         numeri[r][c] = value;
42:     }
43:
44:     private boolean existsDuplicate(int[] n) {
45:         //System.out.print("Controllando:");
46:         //for (int i=0; i<n.length; i++) System.out.print(" "+n[i]);
47:         //System.out.println("");
48:         boolean flag = false;
49:         for (int i=0; i<n.length; i++) {
50:             if (n[i]==0) continue;
51:             for (int j=0; j<n.length; j++) {
52:                 if (n[j]==0) continue;
53:                 if ((n[i] == n[j]) && (i != j)) flag=true;
54:             }
55:         }
56:         return flag;
57:     }
58:
59:     public boolean isLegal() {
60:         boolean flag = true;
61:         for (int i=0; i<size; i++) {
62:             //Check column i
63:             if (existsDuplicate(numeri[i])) {
64:                 //System.out.println("Riga "+i);
65:                 return false;
66:             }
67:             //Check row i
68:             int[] row = new int[size];
69:             for (int j=0; j<size; j++) {
```

```
70:         row[j]=numeri[j][i];
71:     }
72:     if (existsDuplicate(row)) {
73:         //System.out.println("Colonna "+i);
74:         return false;
75:     }
76: }
77: //Check subsquares
78: for (int s1=0; s1<sqrsize; s1++) {
79:     for (int s2=0; s2<sqrsize; s2++) {
80:         int[] subsquare = new int[size];
81:         int k=0;
82:         for (int i=0; i<sqrsize; i++) {
83:             for (int j=0; j<sqrsize; j++) {
84:                 subsquare[k++]=numeri[s1*sqrsize+i
][s2*sqrsize+j];
85:             }
86:         }
87:         if (existsDuplicate(subsquare)) {
88:             //System.out.println("Sottomatrice "+s1+"
"+s2);
89:             return false;
90:         }
91:     }
92: }
93: return true;
94: }
95:
96: public int howMany() {
97:     int c=0;
98:     for (int i=0; i<size; i++) {
99:         for (int j=0; j<size; j++) {
100:             if (numeri[i][j] != 0) c++;
101:         }
102:     }
103:     return c;
104: }
105:
106: public boolean isComplete() {
107:     return (this.howMany() == size*size);
108: }
109:
110: public Coordinates getEmptyCell() {
111:     for (int i=0; i<size; i++) {
112:         for (int j=0; j<size; j++) {
113:             if (numeri[i][j] == 0) {
114:                 Coordinates point = new Coordinates(i, j);
115:                 return point;
116:             }
117:         }
118:     }
119:     return null;
120: }
121:
122: public void print() {
123:     int c=0;
124:     System.out.println("solution: "+this);
125:     for (int i=0; i<size; i++) {
126:         for (int j=0; j<size; j++) {
127:             System.out.print(" "+numeri[i][j]);
128:         }
129:         System.out.println("");
130:     }
131:     System.out.println("-----");
132: }
133: }
```

```
1: package it.unisa.di.aa.sudoku;
2:
3: public class Sudoku {
4:     static int[][] n4 = { {0,0,0,0}, {0,0,0,0}, {0,0,0,0}, {0,0,0,0} };
5:
6:     static int[][] n9 = {
7:         {4,0,0,9,0,1,7,2,0},
8:         {0,0,0,0,0,0,0,0,0},
9:         {0,0,0,0,0,0,0,0,0},
10:        {0,0,0,0,0,0,0,0,0},
11:        {0,0,0,0,0,0,0,0,0},
12:        {0,0,0,0,0,0,0,0,0},
13:        {0,0,0,0,0,0,0,0,0},
14:        {0,0,0,0,0,0,0,0,0},
15:        {0,0,0,0,0,0,0,0,0} };
16:
17:
18:     static int size = 9;
19:
20:
21:     public static void main(String[] args) {
22:         // TODO Auto-generated method stub
23:
24:         Solution sol = new Solution(n9,size,0);
25:         Solution sol2 = recursion(sol);
26:
27:         if (sol2 != null) {
28:             sol2.print();
29:             System.out.println("Solution is: "+sol2.isLegal());
30:         }
31:         else {
32:             System.out.println("No solution");
33:         }
34:     }
35:
36:
37:     private static Solution recursion(Solution s) {
38:         int rl = s.getRecursionLevel();
39:         System.out.println("Recursive level="+rl);
40:         if (!s.isLegal()) {
41:             //System.out.println("s is not legal");
42:             return null;
43:         }
44:         if (s.isComplete()) return s;
45:         //recursion
46:         //find an empty cell
47:         Coordinates point = s.getEmptyCell();
48:         if (point == null) {
49:             System.out.println("point is null. Cannot be!");
50:         }
51:         //fill all possible values
52:         for (int v=1; v<size+1; v++) {
53:             //System.out.println("iteration v="+v);
54:             //Il metodo clone di Java non funziona!!!
55:             Solution news= new Solution(s.getNumeri(),size,rl+1);
56:             news.setCell(point.getR(),point.getC(),v);
57:             Solution r = recursion(news);
58:             if (r != null) return r;
59:         }
60:         System.out.println("Tried all possibilities. Returning null");
61:         return null;
62:     }
63: }
64:
```