Sudoku (数独)

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

Sudoku Puzzle

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

Approach

We will represent the Sudoku 9*9 grid, with a 9*9 2D matrix. We will represent the empty cells with digit 0, in the 2D matrix.

Start filling the empty cells column by column, starting from cell(0,0), e.g. first fill all the empty cells in column1, then fill all the empty cells in column2 and then fill all the empty cells in column3 and so on.

While placing a number in an empty cell, we will first try digit 1, if its not valid due to Sudoku constraints , then we will try digit 2, if even digit 2 is not valid due to Sudoku constraints , then we will try digit 3 and so on. If we are not able to place any of the digits from 1 to 9, then we will backtrack.

While placing a number in a cell, we will always make sure we follow the 3 constraints of Sudoku.

1. If a row already contains the same number we can not place the same number in that row again
2. If a column already contains the same number we can not place the same number in that column again
3. If a 3*3 subgrid(box) already contains the same number we can not place the same number in that grid again

Implementation

```
public class App {

	public static void main(String[] args) {

		int sudokuMatrix[][] =		{  {5, 3, 0,  0, 7, 0,  0, 0, 0},
		                               {6, 0, 0,  1, 9, 5,  0, 0, 0},
		                               {0, 9, 8,  0, 0, 0,  0, 6, 0},

		                               {8, 0, 0,  0, 6, 0,  0, 0, 3},
		                               {4, 0, 0,  8, 0, 3,  0, 0, 1},
		                               {7, 0, 0,  0, 2, 0,  0, 0, 6},

		                               {0, 6, 0,  0, 0, 0,  2, 8, 0},
		                               {0, 0, 0,  4, 1, 9,  0, 0, 5},
		                               {0, 0, 0,  0, 8, 0,  0, 7, 9} };

		Sudoku sudoku = new Sudoku(sudokuMatrix);
		sudoku.solveSudoku();
	}
}
```

```java
public class Sudoku {

	private int[][] sudokuMatrix;

	public static final int BOARD_SIZE = 9;
	public static final int BOX_SIZE = 3;
	public static final int MIN_NUMBER = 1;
	public static final int MAX_NUMBER = 9;


	public Sudoku(int[][] sudokuMatrix) {
		this.sudokuMatrix = sudokuMatrix;
	}

	public void solveSudoku() {

		if (solve(0, 0)){
			printResult();
		} else {
			System.out.println("No feasible solution found...");
		}
	}

	...
}
```

```
private boolean solve(int rowIndex, int columnIndex) {

		// If we filled all the rows of a column, Move to the next column
		if( rowIndex == BOARD_SIZE){
			columnIndex = columnIndex + 1;
			// Base case
			if(columnIndex == BOARD_SIZE) {
				return true;
			}
			// If not the base case, we still need to fill remaining columns
			// So set the rowIndex to zero, for the next column
			rowIndex=0;
		}


		if ( sudokuMatrix[rowIndex][columnIndex] != 0 ) {
			// skip filled cells
			return solve(rowIndex + 1, columnIndex);
		}

		for (int number = MIN_NUMBER; number <= MAX_NUMBER; number++) {
			if ( isValid(rowIndex, columnIndex, number) ) {
				sudokuMatrix[rowIndex][columnIndex] = number;
				if ( solve(rowIndex + 1, columnIndex) ) {
					return true;
				}
			}
		}

		//  BACKTRACK !!!
		sudokuMatrix[rowIndex][columnIndex] = 0;
		return false;
}
```

```java
private boolean isValid(int rowIndex, int columnIndex, int number) {

		// check if the same row alreday have that same digit
		for (int i = 0; i < BOARD_SIZE; i++)
			if ( sudokuMatrix[rowIndex][i] == number )
				return false;

		// check if the same column alreday have that same digit
		for (int j = 0; j < BOARD_SIZE; j++)
			if ( sudokuMatrix[j][columnIndex] == number )
				return false;

		// check if the same subgrid(box) alreday have that same digit
		int boxRowOffset = (rowIndex / 3) * BOX_SIZE;
		int boxColumnOffset = (columnIndex / 3) * BOX_SIZE;

		for (int i = 0; i < BOX_SIZE; i++)
		  for (int j = 0; j < BOX_SIZE; j++)
		   if (sudokuMatrix[boxRowOffset + i][boxColumnOffset + j] == number)
		                   return false;

		return true;
}




private void printResult() {
		for (int i = 0; i < BOARD_SIZE; i++) {

			if(i % 3 == 0) System.out.println(" ");

			for (int j = 0; j < BOARD_SIZE; j++) {

				if (j % 3 == 0) System.out.print(" ");
				System.out.print(sudokuMatrix[i][j]+" ");
			}

			System.out.println(" ");
		}
}
```