Maze Solving (Path Finding, Rat in a Maze)

Approach

- Represent the maze with a 2D matrix. Start from top left square , cell(0, 0) . Try to move to the square on the right, if it's not possible to move to the square on the right because there is obstacle, try to move down. Keep doing this until we reach to the bottom right square, cell(N-1, N-1), where N is the size of the maze. On making a move to valid cell(row, column) mark the value in solutionMatrix[row][column] =1

- If we reach a dead end, a cell(row, column) from where we can neither go right nor down then backtrack (mark the value in solutionMatrix[row][column] = 0 and return false)

Implementation

```
public class App {

	public static void main(String[] args) {

		int maze[][] =  { { 1, 1, 1, 1, 1},
				  { 1, 0, 0, 1, 0},
				  { 0, 0, 0, 1, 0},
				  { 1, 1, 1, 1, 1},
				  { 1, 1, 1, 0, 1}
				};

		Maze mazeInstance = new Maze(maze);
		mazeInstance.solveMaze();

	}
}
```

```
public class Maze {

	private int[][] maze;
	private int[][] solutionMatrix;
	private int mazeSize;

	public Maze(int[][] maze) {
		this.maze = maze;
		this.mazeSize = maze.length;
		this.solutionMatrix = new int[this.mazeSize][this.mazeSize];
	}

	public void solveMaze() {
		if (solve(0, 0) ) {
			printResult();
		} else {
			System.out.print("No feasible solution found...");
		}
	}

	...
}
```

```
public boolean solve(int x, int y) {

		if ( checkBaseCase(x, y)) {
			return true;
		}

		if ( isValidMove(x, y) && this.maze[x][y] == 1 ) {

			solutionMatrix[x][y] = 1;

			// Try going forward
			if (solve(x, y+1)){
				return true;
			}

			// Try going downward
			if (solve(x+1, y)) {
				return true;
			}

			// BACKTRACK !!!
			solutionMatrix[x][y] = 0;
		}

		return false;
}


public boolean checkBaseCase(int x, int y){

  if (x == this.mazeSize - 1 && y == this.mazeSize - 1 && this.maze[x][y] == 1) {
		solutionMatrix[x][y] = 1;
		return true;
  }

  return false;
}




public boolean isValidMove(int x, int y) {
	if( x<0 || x >= this.mazeSize || y<0 || y >= this.mazeSize) {
		return false;
	 }
	return true;
}
```

```
public void printResult() {
	for (int i = 0; i < this.mazeSize; i++) {
		for (int j = 0; j < this.mazeSize; j++){
			if( this.solutionMatrix[i][j] == 1 ){
					System.out.print(" * ");
			 } else {
					System.out.print(" - ");
			}
		}
		System.out.println();
	}
}
```

Note : In a recursive function always make sure your code never fall into an infinite recursion. If your code results in an infinite recursion, you will get a StackOverflowError at runtime, because the amount of memory allocated for call stack will never be enough.