CS1020 Take-home Lab #2
Exercise #3: Gold Hunters

http://www.comp.nus.edu.sg/~cs1020/3_ca/labs.html

Objectives:

1. Using OOP concepts
2. Using two-dimensional arrays

Task statement:

[This is adapted from Sit-in lab #1 question in AY2012/2013 semester 2.]

(Note that unless otherwise stated, you may assume that all input data are valid and hence there is no need for you to perform input data validation.)

A group of gold hunters chanced upon a treasure map showing the way to a mysterious location which is said to be filled with gold smuggled there during World War II. After a hazardous journey hacking through thick jungles and battling man-eating beasts, the hunters finally arrived at the location which turned out to be a large flat valley. Determined to find the gold, they started searching the valley. To their horror, a member was blown up after stepping on a mine. When they looked at their map again, they discovered that the valley is a square grid with markings on some of the cells. They then realized that the marking ☆ denotes the position of a mine.

Map of valley as a grid layout

★		★	
		★	

They were pleasantly surprised to discover that the explosion exposed the gold that was buried in neighboring cells next to the mine. In fact, the amount of gold (in tons) found in a cell is directly proportional to the number of mines surrounding it. That is, if a cell is neighbor to 2 mine cells, then the amount of gold in the cell is 2 tons, if it is neighbor to 3 mine cells, then the amount of gold is 3 tons, etc.

Amount of gold buried at location (0, 4) in a 4×4 grid

0	1	1	1
0	1	★	1
0	1	1	1
0	0	0	0

Gold in map with 1 mine

1	2	1	1
★	2	★	1
1	2	1	1
0	0	0	0

Gold in map with 2 mines

1	2	1	1
★	3	★	1
2	★	2	1
1	1	1	0

Gold in map with 3 mines

One of the gold hunters who is a skilled programmer decided to write a program to compute the amount of gold in each cell. Cells containing mines have no gold.

The map of the valley is represented as an *nRow* × *nCol* grid, with the mines marked on the map.

Input:

The first line contains 2 integers *nRow* and *nCol*, both in the range [1, 100], which represent the number of rows and columns of the grid respectively. Each of the next *nRow* lines contains *nCol* characters, with '*' denoting a mine-filled cell and '.' a mine-free cell. An example is given below:

```
6 6
..*...
.*....
*.*...
....**
...*..
......
```

Output:

The map labeled with the amount of gold (in tons) in each cell. Cells containing mines are still labeled as '*'. The output corresponding to the above input is given below:

```
12*100
2*3200
*3*222
1223**
001*32
001110
```

Number of submissions:

You are given **15** submissions. Only the final submission will be graded.

Skeleton Programs

We provide this skeleton program **MyMap.java** for your use.

```
import java.util.*;

class MyMap {
	private char[][] charMap; // Map containing the characters (labels)
	private int[][] intMap;  // Map containing the gold values
	private int nRow; // number of rows
	private int nCol; // number of columns

	public MyMap() {};

	public MyMap(Scanner sc) {
		nRow = sc.nextInt();
		nCol = sc.nextInt();
		charMap = new char[nRow][nCol];
		intMap = new int[nRow][nCol];
		String line;
		for (int i = 0; i < nRow; i++) {
			line = sc.next();
			for (int j = 0; j < nCol; j++) {
				setLabel(i, j, line.charAt(j));
				setGoldVal(i, j, 0);
			}
		}
	}

	public int getNumRow() {
		return nRow;
	}

	public int getNumCol() {
		return nCol;
	}

	public char getLabel(int row, int col) {
		return charMap[row][col];
	}

	public int getGoldVal(int row, int col) {
		return intMap[row][col];
	}

	public void setLabel(int row, int col, char label) {
		charMap[row][col] = label;
	}

	public void setGoldVal(int row, int col, int val) {
		intMap[row][col] = val;
	}

	// To print the character map
	public void display() {
		// fill in the code
```

```
	}

	// Return true if row, col are within valid range,
	// otherwise return false.
	public boolean indexWithinRange(int row, int col) {
		// fill in the code


	}
}
```

You are to complete the **display()** and **indexWithinRange()** methods. You are NOT to add any other methods into this class, or modify any of the given code.

You are to write a client program **GoldHunter.java** to make use of this **MyMap** class. A skeleton program for **GoldHunter.java** is also provided.

```
import java.util.*;

public class GoldHunter {

	public static void main(String[] args) {

		Scanner sc = new Scanner(System.in);
		MyMap map = new MyMap(sc);

		// To check before anything is done.
		// Remove the statement below after you have tested your program.
		map.display();

		// Fill in the code below


	}

	// Note: You should add some methods in this program.
	//       You should not do all the work in the main() method above.

}
```

The objective of this exercise is work within the constraints of what is given. If you modify the given **MyMap** class or add methods into it, you are violating the objectives of this exercise.

For the program **GoldHunter.java**, to promote modularity, if you find that your nested loop has 4 levels, it is better to create a separate method.