AP Computer Science Homework 17

Due date: Monday, April 18, 2016
Instructor: Mr. Alwin Tareen

Part A: Locating Treasure on a Map

- Submit your Java program `TreasureMap.java` by uploading it to the Web-CAT automated grading platform.

A treasure map is represented as a rectangular grid. Each grid location contains either a single treasure or nothing. The grid is represented using a matrix(two-dimensional array) of `boolean` values. If a cell in the grid contains a treasure then the value `true` is stored in the corresponding matrix location; otherwise, the value `false` is stored.
Consider the following declaration for the `TreasureMap` class.

```
public class TreasureMap
{
    /** Indicates the existence of a treasure at (row, col) */
    private boolean[][] myGrid;

    /** Returns the number of rows in the treasure map.
     *  @return number of rows */
    public int numRows()
    { /* implementation not shown */ }

    /** Returns the number of columns in the treasure map.
     *  @return number of columns */
    public int numCols()
    { /* implementation not shown */ }

    /** Returns true if the cell at location (row, col) contains
     *  a treasure; returns false if location (row, col) is not
     *  within the bounds of the grid, or if there is no treasure
     *  at that location.
     *  @return indicates treasure, no treasure, or out of bounds */
    public boolean hasTreasure(int row, int col)
    { /* to be implemented in part(a) */ }

    /** Precondition: 0 <= row < numRows(); 0 <= col < numCols()
     *  Returns a count of the number of treasures in the cells
     *  adjacent to the location (row, col), horizontally, vertically,
     *  and diagonally.
     *  @return quantity of treasures in adjacent cells */
    public int numAdjacent(int row, int col)
    { /* to be implemented in part(b) */ }

    /** There may be fields, constructors, and methods that are
     *  not shown. */
}
```

For example, suppose that the 6-by-9 grid shown below is a treasure map, where the symbol * in a cell indicates a treasure, and the symbol - indicates no treasure. In this example, `myGrid[2][3]` is `true` and `myGrid[1][3]` is `false`.

	0	1	2	3	4	5	6	7	8
0	-	*	*	-	*	-	*	-	-
1	-	*	-	-	-	-	*	-	-
2	-	*	-	*	*	-	-	*	*
3	*	-	*	-	*	*	-	-	-
4	-	*	-	-	*	-	-	*	-
5	*	-	-	*	-	*	-	-	-

1. Write the TreasureMap method `hasTreasure`, which is described as follows. `hasTreasure` returns `true` if there is a treasure at the location `(row, col)`. If `(row, col)` is not within the bounds of the grid or if there is no treasure at that location, `hasTreasure` returns `false`.

   For example, if `treasureMap theMap` represents the treasure map shown at the beginning of the question, the following table gives the results of several calls to `hasTreasure`.

Function Call	Result
`theMap.hasTreasure(0,2)`	`true`
`theMap.hasTreasure(0,-1)`	`false`
`theMap.hasTreasure(2,3)`	`true`
`theMap.hasTreasure(2,2)`	`false`
`theMap.hasTreasure(4,9)`	`false`

   Complete method `hasTreasure` in the `TreasureMap.java` file.

2. Write the method `numAdjacent`, which is described as follows. `numAdjacent` returns the number of treasures that are adjacent to a given location specified by `row` and `col`. To be adjacent, a treasure must be in one of the(at most) eight cells that border the given location horizontally, vertically, or diagonally; a treasure in the given location does not count as being adjacent.

   For example, if `TreasureMap theMap` represents the treasure map shown above, the following table gives the results of several calls to `numAdjacent`.

Function Call	Result
`theMap.numAdjacent(3,3)`	`5`
`theMap.numAdjacent(2,4)`	`3`
`theMap.numAdjacent(4,7)`	`0`

   In writing `numAdjacent`, you may call `hasTreasure` specified in part (a). Assume that `hasTreasure` works as specified, regardless of what you wrote in part (a).

   Complete method `numAdjacent` in the `TreasureMap.java` file.