Guide Me

Borrowed from hpcodewars and modified

To complete this problem you have been given the completed `Position` class shown below:

```
public class Position
{
    private int x;
    private int y;

    /**
     * Constructor for objects of class Position
     */
    public Position(int x1, int y1)
    {
        x = x1;
        y = y1;
    }

    public int getX()
    {
       return x;
    }

    public int getY()
    {
       return y;
    }

    public String toString()
    {
       return "(" + x + ", " + y + ")";
    }

    public boolean equals(Object obj)
    {
        Position p = (Position) obj;
        return  x == p.getX() && y == p.getY();
    }

    public int hasCode() {  return  x * y; }
}
```

Consider a 2D array `(int[][])` of non-negative `int` values representing the cost of entering each `Position` in the 2D array. A **path** through the 2D array is a sequence of directly adjacent `Positions` (up, left, down, or right - not diagonal) in the 2D array beginning at one `Position` and terminating at a different `Position`.

- **In this problem, you may assume a max of 10 rows and 10 columns.**

In this problem you are to complete the `getMinPathCost and getSummationPath` method in the `GuideMe` class. The cost of a path is the sum of all the cost of each `Position` in the 2D array the path enters. The `getMinPathCost` method computes the minimum cost of all paths from the starting `Position` to the final `Position` (it does not need to return the path). The `getSummationPath` returns a path through the 2D array with the *summation* property.

Guide Me

Borrowed from hpcodewars and modified

The `getMinPathCost` method returns the minimum cost of all paths in the 2D array from a starting `Position` to a final `Position`.

For example, given the following:

```
int[][] grid = { { 1, 7, 9}, {41, 55, 3}, {29, 12, 23}, {11, 2, 4} };
```

The cost of the path through the following `Position`s:

```
new Position(0,0),
new Position(0,1),
new Position(1,1),
new Position(2,1),
```

is: 1 + 7 + 55 + 12 = 75.

The following code shows the results of the `getMinPathCost` method.

The following code	Returns

`int[][] values = { { 1, 17, 19, 13},`
`{41, 55, 3, 18},`
`{29, 22, 23, 1},`
`{31, 20, 17, 4} };`
 	
`GuideMe gm = new GuideMe(values)`	

`gm.getMinPathCost(new Position(0, 0), new Position(2, 1))`	85
`gm.getMinPathCost(new Position(3, 0), new Position(2, 3))`	73

The `getSummationPath` method returns a path through the 2D array with the *summation* property. If no path has the *summation* property, return `null`. A path with the *summation* property is a path which conforms to the following rules:

1. the path begins at a starting `position` (for example: `new Position(0,0)`)
2. the path ends at a final `position` (for example `new Position(3,3)`)
3. as the path progresses, each `position` in the path is directly adjacent (up, left, down, or right) to the previous `position` (no diagonal connections)
4. The path cannot contain loops (that is, it cannot contain the same `Position` more than once), and the value at final `position` (for example: `new Position(3,3)`) is the sum of all the `Position` in the "path" excluding the value at final `position`.

`getSummationPath` should returned a `List` of all `Position` visited on the along the "path" in the order visited. The first value in the `List` should be the starting `Position` and last `Position` in the `List` should be the final `Position`.

An example of the the `getSummationPath` method is on the next page.

Guide Me

Borrowed from hpcodewars and modified

The following code shows the results of the `getSummationPath` method.

`int[][] values1 = { { 12, 26,  7,  31},`
`{101,  8, 61,  44},`
`{ 18, 82, 13, 119},`
`{ 83,  3, 47, 251} };`
 	
`GuideMe gm = new GuideMe(values1)`
 	
`List<Position> ans = gm.getSummationPath(new Position(0, 0),`
`new Position(3, 3))`	

`ans.size()`	`9`
`ans.get(0)`	`new Position(0, 0)`
`ans.get(1)`	`new Position(0, 1)`
`ans.get(2)`	`new Position(0, 2)`
`ans.get(3)`	`new Position(1, 2)`
`ans.get(4)`	`new Position(2, 2)`
`ans.get(5)`	`new Position(2, 1)`
`ans.get(6)`	`new Position(3, 1)`
`ans.get(7)`	`new Position(3, 2)`
`ans.get(8)`	`new Position(3, 3)`