Lab # 6
Dynamic programming

Deadline
See the D2L dropbox

Names of the students

Submission instructions
1) Don't submit your code file; copy and paste your code in the box below. Also, paste the code output as well.
2) If your code doesn't run properly, please copy your code and compile/run the time error you have obtained.

Problem Description:
You are given a rectangular grid of size MxN. The grid is filled with non-negative integers. You can move only down or right from the top-left cell to the bottom-right cell. The objective is to find the path that minimizes the sum of values along its cells.

For example, consider the following 3x3 grid:

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

The path that minimizes the sum of values along its cells is:

Path: [0,0] -> [0,1] -> [0,2] -> [1,2] -> [2,2] Values: 1 -> 2 -> 3 -> 6 -> 9

Sum of values along the path: 1 + 2 + 3 + 6 + 9 = 21

Java Code:
Here is a Java program that solves the above problem using dynamic programming:

```java
public class MinPathSum {
    public int minPathSum(int[][] grid) {
        int m = grid.length;
        int n = grid[0].length;
        int[][] dp = new int[m][n];
```

```
        dp[0][0] = grid[0][0];

        for (int i = 1; i < m; i++) {
            dp[i][0] = dp[i - 1][0] + grid[i][0];
        }

        for (int j = 1; j < n; j++) {
            dp[0][j] = dp[0][j - 1] + grid[0][j];
        }

        for (int i = 1; i < m; i++) {
            for (int j = 1; j < n; j++) {
                dp[i][j] = Math.min(dp[i - 1][j], dp[i][j - 1]) + grid[i][j];
            }
        }

        return dp[m - 1][n - 1];
    }

    public static void main(String[] args) {
        int[][] grid = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        MinPathSum solution = new MinPathSum();
        System.out.println(solution.minPathSum(grid));
    }
}
```

This program uses a 2D array **dp** to store the minimum sum of values along the path from the top-left cell to each cell in the grid. The **minPathSum** method initializes the first row and first column of the **dp** array and then fills in the remaining values using dynamic programming.

The output of this code will be,

```
21
```

This is the minimum sum of values along the path from the top-left cell to the bottom-right cell of the **grid**. The **minPathSum** method computes this value using dynamic programming and returns it. In the **main** method, this value is printed to the console using the **System.out.println** method.

The task for Students:

Modify the above program to print out the path that minimizes the sum of values along its cells. For example, the modified program should output exactly as shown below. This will demonstrate that you have understood the code. Remember that you cannot hard code the strings expected as outputs. If you have any questions, please let me know. You need to modify the given code and do not write the code from scratch

Consider that someone has written the code before you and your task is to modify the code and change the code output.

Expected output

Path: [0,0] -> [0,1] -> [0,2] -> [1,2] -> [2,2]

Values: 1 -> 2 -> 3 -> 6 -> 9

Sum of values along the path: 1 + 2 + 3 + 6 + 9 = 21

Your code

```
import java.util.*;

public class MinPathSum {
    public int minPathSum(int[][] grid) {
        int m = grid.length;
        int n = grid[0].length;
        int[][] dp = new int[m][n];
        int[][] path = new int[m][n];
        dp[0][0] = grid[0][0];
        path[0][0] = -1; // start point
        for (int i = 1; i < m; i++) {
            dp[i][0] = dp[i - 1][0] + grid[i][0];
            path[i][0] = 1; // from up
        }
        for (int j = 1; j < n; j++) {
            dp[0][j] = dp[0][j - 1] + grid[0][j];
            path[0][j] = 2; // from left
        }
        for (int i = 1; i < m; i++) {
            for (int j = 1; j < n; j++) {
                if (dp[i - 1][j] < dp[i][j - 1]) {
                    dp[i][j] = dp[i - 1][j] + grid[i][j];
                    path[i][j] = 1; // from up
                } else {
                    dp[i][j] = dp[i][j - 1] + grid[i][j];
                    path[i][j] = 2; // from left
                }
            }
        }
        // backtrack to find the path
        List<String> pathList = new ArrayList<>();
        int i = m - 1, j = n - 1;
        pathList.add("[" + i + "," + j + "]");
        while (i > 0 || j > 0) {
            if (path[i][j] == 1) {
                i--;
            } else {
                j--;
```

```
            }
            pathList.add("[" + i + "," + j + "]");
        }
        Collections.reverse(pathList);
        // print out the path and values
        System.out.println("Path: " + String.join(" -> ", pathList));
        System.out.print("Values: ");
        int sum = 0;
        for (String p : pathList) {
            String[] ij = p.substring(1, p.length() - 1).split(",");
            int ii = Integer.parseInt(ij[0]);
            int jj = Integer.parseInt(ij[1]);
            sum += grid[ii][jj];
            System.out.print(grid[ii][jj] + " -> ");
        }
        System.out.println("\b\b "); // remove the last arrow
        System.out.println("Sum of values along the path: " + sum);
        return sum;
    }

    public static void main(String[] args) {
        int[][] grid = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        MinPathSum solution = new MinPathSum();
        solution.minPathSum(grid);
    }
}
```

Answer

```
Path: [0,0] -> [0,1] -> [0,2] -> [1,2] -> [2,2]
Values: 1 -> 2 -> 3 -> 6 -> 9
Sum of values along the path: 1 + 2 + 3 + 6 + 9 = 21
```