Problem definition

The Unique Paths problem asks:

Given an m × n grid, a robot starts at the top-left corner (0, 0) and must reach the bottom-right corner (m-1, n-1).
The robot can only move down or right at any step.
The goal is to find the number of unique ways the robot can reach the destination.

Objective - to compute the total number of distinct paths from the top-left to the bottom-right cell, using Dynamic Programming to avoid redundant calculations.

Dynamic Programming Recurrence Relation

Let `dp[row][column]` represent the number of ways to reach cell `(row, column)`.

Then:
`dp[row][column] = dp[row - 1][column] + dp[row][column - 1]`

Because:

- The robot can only come **from the top** `(row - 1, column)` or **from the left** `(row, column - 1)`.
- Base cases:
  - `dp[0][column] = 1` for all `column` (only one way: move right)
  - `dp[row][0] = 1` for all `row` (only one way: move down)

Code

```java
import java.util.Scanner;

public class UniquePaths {
    public static int uniquePaths(int rows, int columns) {
        int[][] dp = new int[rows][columns];

        for (int row = 0; row < rows; row++) {
            dp[row][0] = 1;
        }
        for (int column = 0; column < columns; column++) {
            dp[0][column] = 1;
        }

        for (int row = 1; row < rows; row++) {
            for (int column = 1; column < columns; column++) {
                dp[row][column] = dp[row - 1][column] + dp[row][column - 1];
            }
        }

        return dp[rows - 1][columns - 1];
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("rows: ");
        int rows = scanner.nextInt();
        System.out.print("columns: ");
        int columns = scanner.nextInt();
        scanner.close();
        System.out.println("Unique Paths: " + uniquePaths(rows, columns));
    }
}
```

Example table

row\column	**0**	**1**	**2**	**3**
0	1	1	1	1
1	1	2	3	4
2	1	3	6	10

The number of unique paths for a 3×4 grid = **10**

Time and space complexity analysis

Time complexity - `O(rows x columns)`. Because each cell is filled once.
Space complexity - `O(rows x columns)`. Because 2D DP array is used.