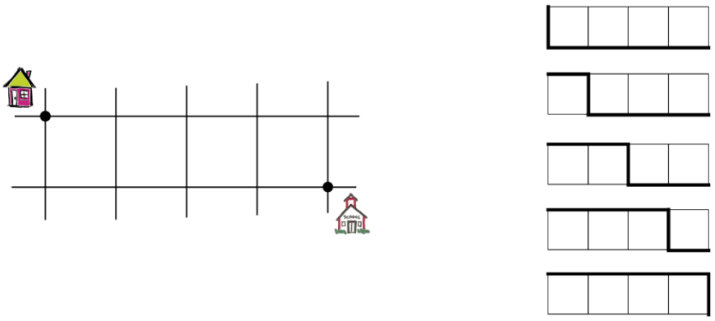
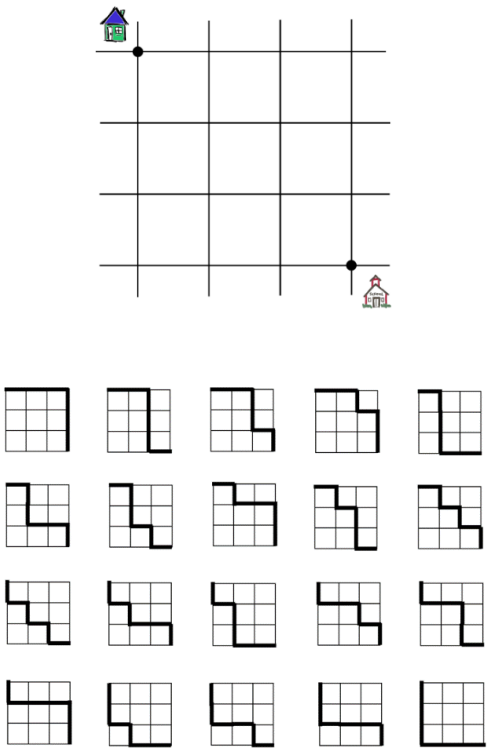
Assignment overview:

Young Johnny lives 5 blocks away from the schoolhouse as shown in the diagram below. When he walks to school he has a choice of five different routes he can take:



Young Alice lives just 6 blocks from the same school, but she can walk **20** different ways:



These numbers do not bother with picky details like Johnny and Alice going back and forth across the street because of people not practicing proper physical distancing. Fortunately, you don't need to worry about that, either.

The recursive formula

Define *SW(m, n)* as the number of different ways that someone can walk *m* blocks down and *n* blocks over on a grid of city streets. There is no walking up or left. The starting point for a walk is (0, 0). Then *SW(1, 4)* = 5 is the number of ways Johnny can walk to school, and *SW(3, 3)* = 20 is the number of ways that Alice can walk to school.

SW(0,0) is equal to 1. There is exactly 1 way to "go" to where you started: do nothing.

In fact, for any points along the topmost (horizontal) street or the leftmost (vertical) street, there is only one way to walk there: a straight line from the starting point.

Key observation: For all other grid points, in order to arrive there, the last other point you walk through must be either the point immediately above, or the point immediately to the left. These two sets of paths have no overlap, because all of the paths in one set have a vertical street for their last segment, but all of the paths in the other have a horizontal street last.

Thus the total number of ways you can arrive to a point is equal to the number of ways you could have arrived in the grid point immediately above *plus* the number of ways you could have arrived in the grid point immediately to the left.

Thus the function *SW(m, n)* can be expressed recursively as follows:

If m = 0: *SW(0, n)* = 1 (walking *n* blocks straight to the right)

If n = 0: *SW(m, 0)* = 1 (walking *m* blocks straight down)

If *m, n* are both > 0: *SW(m,n) = SW(m-1, n) + SW(m, n-1)*

Part 1—Calculate SW(m,n) recursively

Write a function `SW_Recursive(m, n)` that calculates *SW(m, n)* recursively based on the definition given above.

In your `main()`, write a loop that uses `SW_Recursive(m, n)` to calculate and print *SW(i, i)* for successive values of i starting from 0 to at least 15. Use `System.currentTimeMillis()` to measure the execution time of your function, and report it along with the function's return value.

Here are the correct values for *SW(i, i)* up to *i* = 17 (exact running times will obviously vary according to your own programming environment and hardware):

```
SW_Recursive(0,0) = 1, time is 0 ms
SW_Recursive(1,1) = 2, time is 0 ms
SW_Recursive(2,2) = 6, time is 0 ms
SW_Recursive(3,3) = 20, time is 0 ms          ← Young Alice, 20 different walks!
SW_Recursive(4,4) = 70, time is 0 ms
SW_Recursive(5,5) = 252, time is 0 ms
SW_Recursive(6,6) = 924, time is 0 ms
SW_Recursive(7,7) = 3432, time is 0 ms
SW_Recursive(8,8) = 12870, time is 0 ms
SW_Recursive(9,9) = 48620, time is 1 ms
SW_Recursive(10,10) = 184756, time is 1 ms
SW_Recursive(11,11) = 705432, time is 3 ms
SW_Recursive(12,12) = 2704156, time is 12 ms
SW_Recursive(13,13) = 10400600, time is 35 ms
SW_Recursive(14,14) = 40116600, time is 136 ms
SW_Recursive(15,15) = 155117520, time is 485 ms
SW_Recursive(16,16) = 601080390, time is 2094 ms
SW_Recursive(17,17) = 2333606220, time is 8118 ms
```

Part 2—Calculate SW(m,n) with Dynamic Programming

Write a second function `SW_DynamicProg(m, n)` that also calculates *SW(m, n)*. This function will use Dynamic Programming, but *not* recursion. Use an array to store values of *SW(m, n)* so that they can be used in subsequent calculations. ArrayList or other collection types are OK if you wish, but IMHO a plain 2D array is the easiest.

In your `main()`, write a loop that calculates and prints `SW_DynamicProg(i, i)` for successive values of i from 0 to at least 30. Use `System.currentTimeMillis()` to measure the execution time of your function, and report it along with the function's return value.

Here are some more numbers for you to check your output. Note also that these calculations are *FAST* compared to the recursive version:

```
...(starting from SW(0,0), all the same numbers as before, and)...
SW_DynamicProg(15,15) = 155117520, time is 0 ms
SW_DynamicProg(16,16) = 601080390, time is 0 ms
SW_DynamicProg(17,17) = 2333606220, time is 0 ms
SW_DynamicProg(18,18) = 9075135300, time is 0 ms
SW_DynamicProg(19,19) = 35345263800, time is 0 ms
SW_DynamicProg(20,20) = 137846528820, time is 0 ms
SW_DynamicProg(21,21) = 538257874440, time is 0 ms
SW_DynamicProg(22,22) = 2104098963720, time is 0 ms
SW_DynamicProg(23,23) = 8233430727600, time is 0 ms
SW_DynamicProg(24,24) = 32247603683100, time is 1 ms
SW_DynamicProg(25,25) = 126410606437752, time is 0 ms
SW_DynamicProg(26,26) = 495918532948104, time is 0 ms
SW_DynamicProg(27,27) = 1946939425648112, time is 0 ms
SW_DynamicProg(28,28) = 7648690600760440, time is 0 ms
SW_DynamicProg(29,29) = 30067266499541040, time is 0 ms
SW_DynamicProg(30,30) = 118264581564861424, time is 0 ms
```

How far can you go before you have "long overflow"?

How, what, and when to submit:

Please submit the following to the dropbox on Learning Hub:

- Java file(s) containing your source code

Please do not zip or compress your submission (tempting though it may be).

This lab is worth 20 points. 4 to 5 of these points will be allocated for coding style as mentioned in the guidelines given for Lab 2.

It is due at midnight Sunday November 28.

Bonus challenge #1

Ensure that your program can (correctly) calculate all the way up to *SW(37, 37)*.

(This is not required for the assignment; it's just for a small extra challenge.)

Bonus challenge #2

Ensure that your program can calculate *SW(3737, 3737)* (and finish within a small number of seconds). If you attempt this challenge, be sure to let me know so that I will know to test your code for it.

(This is also not required, just for a bit more of a challenge than Bonus Challenge #1.)

Bonus challenge #3

What is the very largest *SW(x, x)* that you can calculate? How long did it take your program to calculate it?